zk-Cinema: Proving Video Provenance in Zero Knowledge

Alexander Frolov
University of Maryland
USA

Jianfeng Guo
Stanford University
USA

Xinyi Zhao
Stanford University
USA

Trisha Datta
Stanford University
USA

Dan Boneh
Stanford University
USA

Ian Miers
University of Maryland
USA

Abstract

Video provenance is an important problem on the modern internet. In response, the Coalition for Content Provenance and Authenticity (C2PA) has developed a standard for verifying video and image provenance where cameras sign captured videos with an on-device secret key. Since videos are generally edited and resized before being posted, the C2PA signature from a camera cannot be used as is to verify provenance of published videos. Prior work has developed zero-knowledge techniques for verifying provenance of edited images and videos. In this work, we develop new efficient techniques for producing such zero-knowledge proofs. First, we show how to represent common video edits as matrix multiplications in a form that is particularly friendly for zero-knowledge provers and enables a number of optimizations. Second, we develop a SNARK-friendly video representation, which we call SFVR, that reduces prover work for video editing. Third, we design new efficient methods for incorporating signed data into a SNARK proof. To evaluate our designs, we built an end-to-end system for proving edits to a signed video. In our end-to-end system, we optimize the NeutronNova folding scheme for high-arity folding. To scale the size of our NeutronNova proofs, we implement a "Read-Write Streaming" version of NeutronNova to take advantage of high-performance storage and parallel computing resources. Our system achieves competitive performance and scale relative to prior work.

Keywords

video provenance, zero-knowledge proofs, zk-SNARKs, streaming SNARKs, digital signatures

1 Introduction

The advent of generative AI has made verifying the origin of digital videos much harder. This greatly increases the threat of misinformation and disinformation in news [22, 23, 28, 51, 57]. See [24] for a longer discussion. In response, a content provenance standard created by the Coalition for Content Provenance and Authenticity (C2PA) [17] proposes that every camera have a certified digital signing key ([1, 2, 58]) that it can use to sign all videos it takes along with some metadata (e.g., timestamp, aperture, focal length). Even images and videos generated by some generative AI companies, like OpenAI, are now being produced with C2PA-compliant signatures [5].

Per the C2PA, verifying the metadata of a video or photo amounts to verifying the camera's digital signature on the video or photo. However, photographic content is often edited before being published online, especially in newsrooms where video may be resized, trimmed, and parts of it may be blurred for privacy (see [3] for a list of permitted edits by the Associated Press). Because digital signatures can only be verified using the exact data that was signed, an end user who only has access to the edited video or photo cannot verify the accompanying C2PA signature. To address this issue, the C2PA proposes that all edits be made by a trusted program that stores a secret signing key and signs lists of the edits it applies. This is undesirable because editors and end users must now trust the editing software and the security of its signing key.

Prior work [24–26, 35, 40, 43, 53, 76] has proposed replacing trusted editing software with a succinct zero-knowledge proof (zk-SNARK). A zk-SNARK proof for a public statement convinces the verifier that the statement is true while revealing nothing about the corresponding witness beyond what is implied by the statement. In the systems cited above, a news editor can use a zk-SNARK to prove that a published image is the result of applying specific edits to an original C2PA-signed image. Crucially, zk-SNARKs are complete, meaning that verification will pass for true statements, and knowledge-sound, meaning verification will fail if the prover does not have a valid witness (i.e., an original signed image). These properties mean that the verifier does not need to trust the prover. Thus, unlike the C2PA's trusted software approach, using a zk-SNARK introduces no additional trusted parties for verifying image/video provenance, and preserves end-to-end security.

In this work, we address some limitations of prior work on building zk-SNARKs for video editing.

First, verifying video editing proofs is prohibitively expensive in prior work. To our knowledge, Eva [76] is the only prior work that generates zk-SNARKs for video edits. For longer videos, verifying Eva's proofs requires almost twice the amount of time as playing the edited video. Applying prior works on image editing [25, 26, 35] to video would also lead to a relatively expensive verification step because they require the verifier to apply a SNARK-friendly hash to the edited videos, or other expensive work. Our system substantially reduces verification time by eliminating the need for the verifier to evaluate a SNARK-friendly hash of the edited video.

Second, in all prior work, proving image and video edits requires cryptographic work proportional to the uncompressed size of the image or video, which limits prover efficiency. We introduce a SNARK-friendly video representation (SFVR) that allows the prover to do work proportional to the differences between consecutive video frames, which is often smaller than the frames themselves.

Third, prior work for video edits only implements simple transformations. Eva [76] only implements operations which independently edit each pixel of an image, like grayscale conversion or image brightening. In contrast, we generate video editing proofs

for more complex transformations such as (Gaussian) blurring, resizing, and sharpening. We express each transformation as a matrix multiplication and describe new techniques to efficiently verify these transformations in a zk-SNARK. We show that our implementations are close to the implementations in FFmpeg, a standard video editing program.

Finally, using a zk-SNARK to prove that edits were correctly applied to a signed video requires generating a proof for a very large statement. We extend the algorithmic results of Scribe [13] and NeutronNova [44] to build a proof system that can prove large uniform statements quickly on a CPU, even when the size of the statement being proved exceeds the machine's RAM capacity.

Our system. Here we present zk-Cinema, a system for using zk-SNARKs to prove video provenance for losslessly compressed videos. As in prior work, in zk-Cinema, a digital signature from a C2PA-enabled camera on a video V_0 serves as our root of trust. An editor applies some transformation f to produce an edited video V_1. Then, a prover (which may be a different entity from the editor) produces a proof π showing that it knows an original unedited video which has a valid C2PA signature and that the edited video V_1 is the result of applying f to that video. Verifiers can then verify this zk-SNARK. We add two optional features to this architecture. First, our system supports an "assistant," which can be used by a computationally weak party, like a camera, to verifiably outsource the computation of a SNARK-friendly signature on the captured media. Second, while our system has lower proof verification costs than prior work, verification can still be somewhat costly. To alleviate this issue we propose the use of a transparency log: a log that lists hashes of videos for which the proof has already been verified. A weak client that needs to validate an edited video can consult the log to check that multiple parties have already successfully verified the proof for this video. Of course, any client can choose to ignore the log and verify the proof for itself.

We next give a high-level overview of the main technical components of zk-Cinema.

Efficiently realizing real-world video edits in SNARKs. To prove that a public edited video is the result of applying some authorized edits to a signed video, we must design circuits that apply edits to videos. We represent common video edits (e.g., blurring, resizing, or sharpening) as matrix multiplications and develop techniques to efficiently verify these multiplications inside a zk-SNARK circuit. Notably, we can represent many of the edits in the Associated Press' guidelines for allowable photo edits [3] as matrix multiplications or redactions, so our system can cover the set of use cases for video journalism [35]. We introduce a new variant of Freivald's algorithm [29] for verifying a double-sided matrix multiplication. We discuss this approach in Section 5 and present our experimental results in Section 9.1. We show that this double-sided Freivald's gives a significant speedup. Moreover, when an editor applies the same transformation to many frames, we can further decrease the verifier's work by treating the video as an order-3 tensor (i.e., a three dimensional matrix) and use a generalization of Freivald's algorithm to such tensors.

SNARK-Friendly Video Representations. A crucial observation of zk-Cinema is that the prover and verifier need not operate over the existing compressed video. We introduce SFVR, a SNARK-friendly compressed representation for video, described in Section 6. SFVR enables the prover to operate on a compressed format optimized to save in-circuit costs. In zk-Cinema, a camera can sign both the captured compressed video and the SFVR of that video. The prover then proves that applying the claimed transformation to the SFVR of the unedited video results in an encoding of the final edited video.

SFVR leverages the natural redundancy in videos, namely that successive frames tend to be close in Hamming distance. The format supports efficient SNARK proofs of edits which correspond to linear transformations. We show that the prover only needs to do work proportional to these differences instead of work proportional to the size of the entire video. This is quite different from standard video compression where the goal is to reduce overall file size. Instead, the goal of SFVR is to reduce the time to generate video editing proofs.

Implementation via high-arity folding. We implement a modified version of the NeutronNova folding scheme [44] for proving video edits. NeutronNova is a high-arity folding scheme (see Appendix A.4 for a formal definition). High-arity folding schemes enable efficiently folding together more than two circuit instances, where normal folding schemes only fold together two instances. As observed in previous works [26, 76], folding is well-suited to image/video editing because of the inherent parallel structure in these tasks. To scale the number of instances folded by NeutronNova past the RAM limits of a single machine, we implement a "Read-Write Streaming" version of the NeutronNova prover, which uses solid-state storage rather than RAM to store prover state. Our system builds on Scribe [13] which introduced this paradigm. We scale their approach to use more threads and show how to apply their techniques to a folding-based prover. We also introduce additional folding-specific optimizations to their approach. This "streaming" prover implementation is at most 20% slower than a direct RAM-based prover. However, for larger problem instances that a RAM-based prover cannot handle, the streaming prover achieves improved prover throughput because of better parallelism and amortization of fixed costs. We are able to prove editing of 2 minute, 720p videos in a single folding instance, folding up to 10,800 instances of circuits with nearly 2^{23} constraints.

SNARK-friendly schemes for authenticating data. VerITAS [24] introduced a scheme where a computationally strong signer signs a polynomial commitment to a photo. This eliminates the need for signature verification and hashing inside the SNARK circuit (signature verification is done outside of the SNARK circuit). This scheme is not suitable for cameras because they cannot efficiently compute a polynomial commitment to large degree polynomial. The VerITAS scheme is also tailored to the Plonk proving system [31]. We extend the VerITAS method in three ways for the C2PA setting. First, we generalize the method beyond Plonk. In Section 7, we show that a signed polynomial commitment can be used to eliminate in-circuit signature verification in any SNARK system whose circuits can access random challenges. We give an especially efficient instantiation for the Spartan proof system [63]. Second, we show how to verifiably outsource the computation of a polynomial commitment from a weak signer to an untrusted assistant, which enables any

signer, regardless of its computational power, to take advantage of this scheme. Third, we show that signed polynomial commitments support especially efficient proof systems for redactions, such as trimming a video file. We evaluate these constructions in Section 9.3.

Addressing high verification costs. In the prior state of the art [76], verifying a proof of video edits takes almost twice as long as viewing the underlying video.[1] Some prior works on image editing [24, 35] also have relatively high proof verification costs. In contrast, in our scheme, proofs are efficient to verify, taking 26 seconds to verify a proof of editing for a 2-minute 720p video. Our system achieves such low verification costs because it treats a full edited video as a public output of the SNARK, thereby eliminating the need for the verifier to compute a SNARK-friendly hash of the output, as in prior work. Despite these improvements, it might still be impractical for every client to verify every video proof on its own. Instead, we propose a privacy-preserving transparency log that records that the proof for a video has already been verified by others, as discussed earlier in the introduction.

Our contributions. To summarize, our contributions are:

- We show how to implement convolutions (the core operation for many common video edits like blurring or sharpening) more efficiently in SNARK circuits. Our approach is more efficient when applying the same edit to multiple frames, as is common in video editing. Our approach for generating a proof of a moderately complicated convolution applied to a frame is 1.69-9.5× faster than the naive approach in the proof systems tested.
- We introduce a SNARK-Friendly Video Representation (SFVR), which leverages natural redundancy in videos to decouple the prover's workload from the raw video size, significantly reducing proof generation time. Concretely, in our test video suite, this results in a 25.7% average reduction in prover work.
- We implement an optimized version of the NeutronNova folding scheme [44] which uses Read-Write Streaming [13] to decrease the RAM requirements for proving very large statements. We implement several algorithmic and systems-level optimizations and scale the Read-Write Streaming approach to larger thread counts, disk sizes, and circuit sizes than previous work. Our experiments show that the overhead of Read-Write Streaming is less than 20% compared to direct in-memory proving on our machine. However, we stress that our approach actually improves prover throughput in some cases.
- For common proof systems, we show how to lower the cost of verifying a signature in a SNARK circuit using signed polynomial commitments. We also show how to securely outsource the computation of these polynomial commitments from a weak camera to an untrusted server and how to directly prove trimming/redaction operations with signed polynomial commitments.
- We discuss how to address verification costs. First, our use of the NeutronNova proof system supports a lower verification time. Second, we propose using a transparency log so that once enough trusted parties verify the proof for an edited video, other parties can simply look this up in the log.
- Finally, we provide an end-to-end implementation for proving video edits efficiently. We show that our system outperforms all relevant prior works that use CPU-based proving.

[1] Eva [76] splits proof verification into "hashing" and "SNARK verification" steps, but both are required to verify a proof.

2 Preliminaries

We use $[n]$ to denote the set $\{0, \ldots, n-1\}$ and $[a, b]$ to denote the set $\{a, \ldots, b-1\}$. For a vector $m = (m_0, \ldots, m_{n-1})$, let $m[a : b] = (m_a, \ldots, m_{b-1})$. Let $\mathbb{F}_q$ denote a finite field of size q; sometimes we omit the q. Let $r \leftarrow\!\!\$\ S$ denote sampling a value uniformly at random from the finite set S. We use $\mathbb{F}[X_1, \ldots, X_\mu]^{\leq d}$ to denote a polynomial in μ variables of individual degree at most d.

Definition 2.1. *The **multilinear extension** (MLE) [69] of a function $f : \{0,1\}^\ell \to \mathbb{F}$ is the unique ℓ-variate multilinear polynomial $g(x)$ such that $f(x) = g(x)$ for all $x \in \{0,1\}^\ell$. We denote this as $g(x) = \mathsf{MLE}(f(x))$.*

The functions f, g agree on the Boolean hypercube if $f(x) = g(x)$ for all $x \in \{0,1\}^\ell$. We define the MLE of a vector $\vec{x} = (x_0, \ldots, x_{n-1})$ as the MLE of the function that maps the binary representation of i as a vector to x_i for all $i \in [n]$. If the length of $\vec{x}$ is not a power of 2, $\mathsf{MLE}(\vec{x})$ gives the multilinear extension of $\vec{x}$ padded with zeros to make its length a power of 2.

Definition 2.2. *A **signature scheme** [14] is a tuple of* PPT *algorithms* $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *where:*

- $\mathsf{KGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$*, where* pk *is a public verification key and* sk *is a secret signing key.*
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$*, where σ is a signature on message m with private key* sk*.*
- $\mathsf{Vf}(\mathsf{pk}, m, \sigma) \to 0/1$*, where* 1 *means σ is a valid signature on m and* 0 *means it is invalid.*

A signature scheme is secure if it is existentially unforgeable under a chosen message attack [14] (defined in Appendix A.1). To sign data in practice, we first hash the data with a collision-resistant hash and then sign the hash.

We repeat the definitions of polynomial commitment schemes and zk-SNARKs from [24], and we repeat the definition of folding schemes from [44].

Definition 2.3. *A **polynomial commitment scheme** [41] is a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Vf})$ *where:*

- $\mathsf{Setup}(1^\lambda, \ell, d) \to \mathsf{pp}$*, where* pp *are public parameters to commit to a polynomial with $\leq \ell$ variables of individual degree $\leq d$ with security parameter λ.*
- $\mathsf{Commit}(\mathsf{pp}, f, r) \to \mathsf{com}$*, where* com *is a commitment to polynomial $f(x) \in \mathbb{F}[X_1, \ldots, X_\ell]^{\leq d}$ using randomness $r \in \mathcal{R}_{\mathrm{C}}$.*
- $\mathsf{Open}(\mathsf{pp}, f, \vec{x}, r) \to (\pi, y)$*, where π is an opening proof that proves that $f(\vec{x}) = y$.*
- $\mathsf{Vf}(\mathsf{pp}, \mathsf{com}, \vec{x}, y, \pi) \to \{0,1\}$*, where* 1 *implies that π is a valid opening and* 0 *implies π is invalid.*

A secure polynomial commitment scheme must be correct, binding, and hiding (see Appendix A.2 for formal definitions).

Definition 2.4. *A **zk-SNARK** (zero-knowledge succinct non-interactive argument of knowledge) for a relation $\mathcal{R} = (x, w)$ is a triple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ *where:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$, *where* pp *are public parameters.*
- $\mathsf{Prove}(\mathsf{pp}, x, w) \to \pi$, *where* π *shows that* $(x, w) \in \mathcal{R}$.
- $\mathsf{Vf}(\mathsf{pp}, x, \pi) \to \{0, 1\}$, *where* 1 *means that* π *is a valid proof, and* 0 *means* π *is invalid.*

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

Definition 2.5. *Consider a relation* $\mathcal{R}$. *A* ***folding scheme*** *[44] is defined by a tuple of PPT algorithms* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *and a deterministic algorithm* $\mathcal{K}$ *called the generator, the prover, the verifier, and the encoder respectively.*

- $\mathcal{G}(1^\lambda, n) \to \mathsf{pp}$*: takes security parameter* λ *and size parameter* n *and outputs public parameters* pp
- $\mathcal{K}(\mathsf{pp}, s) \to (\mathsf{pk}, \mathsf{vk})$*: takes public parameters* pp *and structure* s *as input and outputs prover key* pk *and verifier key* vk
- $\mathcal{P}(\mathsf{pk}, \{(u_i, w_i)\}_{i\in[m]}) \to (u, w)$*: takes as input public parameters* pp *and* m *instance witness pairs* (u_i, w_i) *and interactively reduces the task of checking* $(\mathsf{pp}, s_1, u_i, w_i) \in \mathcal{R}$ *for all* $i \in [m]$ *to the task of checking* $(\mathsf{pp}, s_2, u, w) \in \mathcal{R}$
- $\mathcal{V}(\mathsf{vk}, \{u_i\}_{i\in[m]}) \to u$*: takes as input public parameters* pp *and* m *instances* $\{u_i\}_{i\in[m]}$ *and interactively reduces the task of checking instances* $u_i \in \mathcal{R}$ *for all* $i \in [m]$ *to the task of checking a new instance* u *in* $\mathcal{R}$

A folding scheme must be complete and knowledge-sound (see Appendix A.4 for formal definitions). As discussed in [45], folding schemes are particularly well-suited for generating zk-SNARK proofs of computations with repeated structure. Prior works [7, 26, 76] use folding schemes to build efficient SNARK provers for real-world computations.

3 Threat model

In the C2PA setting, media produced by a camera is signed using a certified key embedded in the camera. We follow the design of C2PA and assume that the camera is part of the root of trust: the attacker cannot extract the private key nor cause the camera to sign media it did not authentically record.

In the setting of our paper, we add the following parties: a verifier, an editor/prover, and two optional parties: an untrusted assistant server, to handle some cryptographic computations for the camera, and a transparency server that allows clients to verify videos without the cost of verifying the proofs themselves.

A verifier wants to check that a public edited video is the result of applying some public transformation to an original signed video (cropping, blurring, resizing, etc.). These transformations are selected and performed by the editor who wishes to keep the original unedited signed video confidential. The edits are authenticated with a proof possibly produced by a distinct proving party. The editor/prover is untrusted and may try to alter the video content beyond the claimed edits.

The untrusted assistant server computes a polynomial commitment (PCS) to any data it receives (e.g., a video). Our camera can offload PCS computations to this untrusted server, but must verify that the result was computed correctly. We develop protocols for this in Section 7.2.

Finally, the transparency server acts as a transparency log and records the following data for each video: (i) the hash of a (lossily compressed) edited video, (ii) the corresponding losslessly compressed edited videos, (iii) the editing proof proving that the lossless edited video is the result of applying the claimed edits to a C2PA-signed video, and (iv) a list of signatures and identities of verifiers who verified the editing proof. Other verifiers can choose to trust an edited video if the transparency log entry for this video (identified by its hash) contains sufficiently many signatures by trusted parties saying that they verified the editing proof. This greatly reduces the number of verifiers who need to verify the proof for themselves. We stress that a verifier who verifies the editing proof must also certify that errors introduced into the edited video due to lossy compression do not change the semantics of the edited video. They do so by visually comparing the lossless edited video stored on the transparency log with the lossy compressed edited video that is distributed to clients.

3.1 Out of scope threats

We now briefly cover some out-of-scope threats, and refer the reader to [24] for a broader discussion. First, C2PA keys may be improperly issued or stolen. The C2PA specification has provisions for revocation and some deployments use TEEs to secure keys, though key extraction attacks cannot be ruled out. More work is needed here, but this is an orthogonal problem to generating proofs of video edits. A related, out of scope, problem is picture-of-picture attacks, where a genuine camera films a scene rendered on a display. We note that attacks and defenses to these attacks are the subject of active work [56].

We also do not consider adversarial ML attacks [33, 48] where C2PA video, processed with zk-Cinema, is edited specifically to deceive an ML algorithm consuming the final media. For instance, an attacker might try to leverage the small discrepancy our system permits in some cases between captured and edited media to inject adversarial noise. We leave a deeper exploration of ML-focused adversarial attacks to future work (see Section 11).

Finally, while crucial for comprehensive media verification, this paper does not consider end-to-end verification for audio embedded in the video. Authenticating audio manipulations introduces a distinct set of technical challenges and domain expertise, which requires its own dedicated investigation beyond the scope of this paper.

4 System design

This section describes our proposed high-level system design. Figure 1 gives an end-to-end flow for the system. Figure 2 details the prover's internal workflow. We now describe the work performed by each party in the system.

Camera: After capture, the camera sends the raw video V_0 to an assistant server to verifiably obtain a polynomial commitment C using the technique of Section 7.2. The camera then signs the polynomial commitment and (C, σ) serves as a signature for V_0.

Assistant: The assistant is a stateless (powerful) helper which merely computes commitments and openings for any video it receives, as described in Section 7.2.

Editor: The editor decides which editing operations to apply to the video, produces an edited video, and sends the edited video V_1 and a specification of edits performed to the prover.

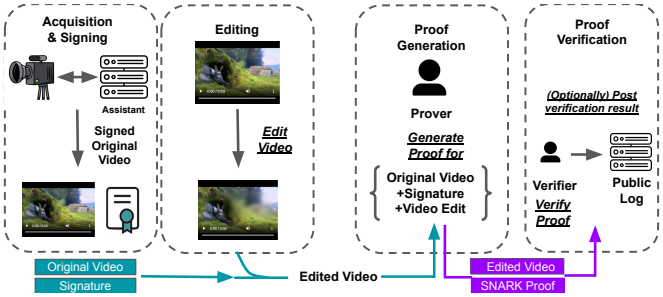

Figure 1: End-to-end workflow for the zk-Cinema pipeline.

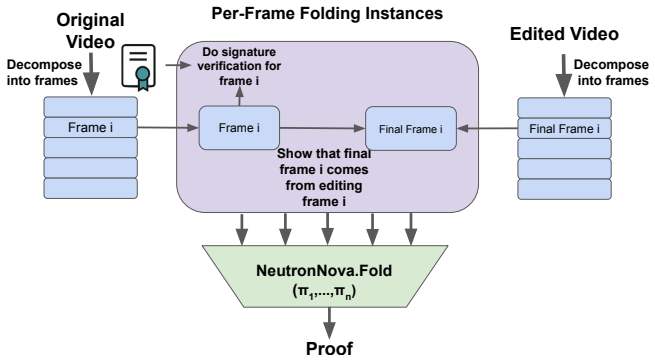

Figure 2: The prover's internal workflow

Prover: These steps are described in Figure 2.

(a) Decompose (V_0, V_1) into two sequences of frames.
(b) For every pair of frames at the same point in the original/edited videos, generate a witness for a NeutronNova instance proof for that frame, and a commitment to this witness for folding. Each circuit shows that: (1) the sequence of edits transforms the original frame i into the final frame i', and (2) the signature verification work for this frame was performed correctly.
(c) Use the NeutronNova folding scheme to fold all the folding instances into a single proof, producing a proof π attesting that *all* frames of V_1 follow from V_0 exactly as claimed.

Verifier/Auditor: On edited clip V_1 and proof π, the verifier:

(a) Decomposes V_1,
(b) Derives the public inputs expected by the circuit
(c) Runs the SNARK verifier and accepts iff the check succeeds.
(d) (Optionally) Posts the result of verification to a public log. Low-power verifiers can audit this log instead.

5 Proving video transformations

In this section, we explain our approach for generating proofs that certain edits were made to the frames of a video. We first discuss how to express common video operations as matrix multiplications. As mentioned previously, many of the allowable edits, as defined by the Associated Press [3], can be expressed as matrix multiplications or redactions, meaning the approaches of this paper are enough to express many relevant video edits. We then show how to efficiently verify these matrix multiplications in a SNARK circuit using a variant of Freivald's algorithm. Lastly, we generalize Freivald's algorithm to work with order-3 tensors rather than matrices, and show how to "batch" multiple proofs together so the cost for verifying a video edit to multiple frames is much less than verifying a video edit for each frame individually. We discuss how these algorithms relate to prior work in Section 10.

5.1 Expressing image transformations as matrix multiplications

In this section, we recall a fact from image processing: common image transformations are based on linear operations and thus can be expressed in terms of matrix multiplications of pixel values. This observation is also made in the concurrent work [35].

A common task in image processing is computing a convolution of the pixels of an image with a fixed kernel (as occurs in blurring or sharpening). The pixel at position x, y in the image f after a discrete convolution with the 2-dimensional kernel ω can be expressed as: $\sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega_{i,j} f_{x-i,y-j}$. Pixels typically have 3 or 4 color channels, and this convolution would be performed separately for each color channel of the image.

Common kernels are ***separable***, meaning that they can be written as the product of two simpler kernels, typically the outer product of 1-dimensional kernels. As a running example, we consider blurring, where the value of each channel of a pixel of an image is set to the weighted average of the pixels in a rectangular neighborhood around that pixel. For example, the following is a simple kernel that averages a pixel's 9 neighbors[2]:

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \times \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{1}$$

Separability allows for more efficient implementations of these convolutions: 1-dimensional convolutions can be applied to the rows of an image and then the columns, which is more efficient than convolution with a 2-dimensional kernel. Libraries like FFmpeg use this approach to improve performance [32].

Applying a convolution to the rows or columns of an image can be expressed in terms of matrix multiplication. For example, for the kernel from (1), "vertically blurring" the columns of an image of height n can be implemented by left-multiplying a matrix of image pixels of one color channel (arranged in the natural way) by the following $n \times n$ matrix[3]:

$$M_{\text{blur},n} = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & \dots \\ 0 & 1/3 & 1/3 & 1/3 & \dots \\ 0 & 0 & 1/3 & 1/3 & \dots \\ 0 & 0 & 0 & 1/3 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2}$$

Blurring the rows of an image can be done via a right multiplication by a similar matrix $M_{\text{blur},n}$, so we can express blurring an $m \times n$ image $\mathcal{I}$ as:

$$\mathcal{I}_{\text{blurred}} := M_{\text{blur},m} \times \mathcal{I} \times M_{\text{blur},n}$$

[2]This is the avgblur kernel in FFmpeg [49] with radius 1.
[3]In this example, we are handling blurring pixels around the edges in the simplest way, though other approaches exist.

This same approach can be used to express image resizing, sharpening and a more complicated "Gaussian blur" as matrix operations. In a Gaussian blur that blurs a neighborhood of $n \times n$ pixels (n is 61 in our implementation[4]), the values of the vertical kernel and the horizontal kernel are vectors containing the weights of a discrete Gaussian with a properly chosen standard deviation [74].

Image resizing can be similarly expressed in terms of matrix multiplication, though it is not exactly a convolution. In particular, a commonly used resizing approach, called bilinear resizing, first resizes a frame along rows and then columns [54]. Every pixel in the resulting image is a linear combination of pixels in the original image. Thus, resizing can also be expressed as a matrix multiplication. Overall, we can reduce any image transformation that can be expressed as a convolution with a separable kernel, as well as a resizing transformation, into left- and right-matrix multiplications.

5.2 Two-sided Freivald's Algorithm

Given a video transformation represented as a sequence of matrix multiplications, we next discuss how to verify such matrix multiplications in a SNARK circuit. We recall a variant[5] of the classic Freivald's Algorithm [29] which is a randomized protocol for verifying matrix multiplication.

In Freivald's Algorithm, the verifier is given matrices A, B, and a claimed product C. For exposition, let the matrices be in $\mathbb{F}^{n\times n}$. To verify their product, the verifier samples a uniformly random vector $r \in \mathbb{F}^n$, computes $(r^T A)B$ and $r^T C$, and checks that these are equal. The verifier's checks take $O(n^2)$ time, which is faster than the $O(n^{2.37})$ time it would take to compute AB directly. The protocol is complete and sound with soundness error $1/|\mathbb{F}|$.

Now, consider using Freivald's algorithm inside of a zk-SNARK circuit. We can further optimize Freivald's Algorithm in the context of zero-knowledge proofs when the matrix A is public. The circuit inside a zk-SNARK can compute $(r^T A)B$, $r^T C$, and check their equality. However, $r^T A$ can be computed outside the circuit and passed in as a public input. This is because $r^T A$ is defined over public data so that the verifier can compute this value on its own. This removes a matrix-vector product from the circuit, decreasing prover work at the cost of an out-of-circuit matrix-vector product for the prover and verifier.

For $n \times n$ matrices, this variant of Freivald's Algorithm requires one out-of-circuit matrix-vector product for the prover and verifier (for $r^T A$), and 2 in-circuit matrix-vector products (to compute $(r^T A)B$ and $r^T C$).

Two-sided Freivald's. We observe that Freivald's Algorithm can be extended to simultaneously verify a left- and right-matrix multiplication, as is needed for verifying a separable filter. Consider verifying a product $ABC = D$ using the following verification algorithm:

$\mathsf{2SidedFreivaldsVerify}\begin{pmatrix} A\in\mathbb{F}^{m\times m}, & B\in\mathbb{F}^{m\times n}, \\ C\in\mathbb{F}^{n\times n}, & D\in\mathbb{F}^{m\times n} \end{pmatrix}$:

1: $r \leftarrow\$ \mathbb{F}^m, \quad s \leftarrow\$ \mathbb{F}^n$

2: Compute $r^T A,\ Cs$

3: **return** $(r^T A)B(Cs) == r^T Ds$

Theorem 5.1. *The two-sided Freivald verifier is complete and sound with soundness error $2/|\mathbb{F}|$.*

To prove the theorem, suppose that $D = ABC + \Delta$ for some non-zero matrix $\Delta \in \mathbb{F}^{m\times n}$. Because the kernel of a non-zero matrix Δ contains at most $|\mathbb{F}|^{n-1}$ vectors we know that $\Pr_s[\Delta \cdot s = 0] \le 1/|\mathbb{F}|$. By the union bound it follows that $\Pr_{r,s}[r^T \cdot \Delta \cdot s = 0] \le 2/|\mathbb{F}|$, which gives an upper bound on the soundness error. If the matrices are defined over a small field $\mathbb{F}$, the checks in the two-sided Freivald's verifier should be iterated or performed in an extension field to achieve an adequate security level.

This two-sided verification approach is considerably more efficient than verifying that $ABC = D$ by running the standard Freivald's Algorithm twice, since computing $(r^T A)B(Cs)$ and $r^T Ds$ requires two matrix-vector products and two inner products, rather than four matrix-vector products.

Now once again, consider applying this algorithm in a zk-SNARK context. As before, when A and C are public matrices, the quantities $r^T A$ and Cs can be computed outside of the SNARK circuit and passed in as public inputs. This approach does not obviously yield a succinct verifier, since the verifier must compute $r^T A$ and Cs, which takes time linear in the size of the matrices A and C. However, in our use case, where A, C represent the application of a small separable kernel to an image, these matrices are sparse, so computing these products takes linear time in the width/height of the image (and thus proportional to the square root of the image size), yielding a sublinear verifier.

Some common video edits involve floating point arithmetic. We opted to represent these operations using fixed point arithmetic, and we defer discussion of considerations around implementing fixed point arithmetic in this context to Section G. This includes rounding, which is also an important part of the transformations we implement.

5.3 Freivald's algorithm for tensors

Our goal is to apply the verification algorithms developed so far to video transformations. Since a video consists of a sequence of multiple frames (images), our prover will be applying the same transformation, such as resizing, to many frames at once. In this section, we show how to batch-verify many transformations at once: when verifying a transformation that is applied to k consecutive frames, our batching scheme decreases the verifier's work by a factor of k, and also decreases the prover's work. This can be viewed as a generalization of Freivald's algorithm to a mode i product between an order 3 tensor and a matrix.

Consider a sequence of frames $F_1, F_2, \ldots, F_k$, to which we want to apply the transformation matrices A_{left},A_{right} to get the edited frames:

$$A_{\text{left}}F_1A_{\text{right}},\ A_{\text{left}}F_2A_{\text{right}},\ \ldots,\ A_{\text{left}}F_kA_{\text{right}}.$$

[4]At the default standard deviation setting in FFmpeg, a radius of 30 is where the contributions of pixels no longer become significant.

[5]This variant left-multiplies by r rather than a right-multiplication.

Now, consider applying the algorithm 2SidedFreivaldsVerify using the same randomness r, s for all k matrix products. Repeating the security analysis of Section 5.2, this achieves a soundness loss of $2k/|\mathbb{F}|$. When sharing the same randomness r, s across multiple instances, we call this approach "batched Freivald's Algorithm."

When implementing batched Freivald's Algorithm to verify k convolutions in a zk-SNARK, the prover can commit to less data: it can commit to the vectors $r, s,\ r^T A,\ As$ once, rather than doing so k separate times. More interestingly, the verifier only needs to derive random challenges r, s and compute the matrix vector products $r^T A$ and As outside of the circuit once, rather than k times. This means that this "batched Freivald's" approach decreases the verifier's work by a factor of k when verifying k convolutions.

This "batched Freivald's algorithm" can be viewed as applying Freivald's algorithm to a mode i product between a third-order tensor and a matrix. For the purposes of this discussion, we are viewing tensors in the sense of machine learning, where they are an arbitrary-dimensional generalization of matrices [61]. This type of product is one way to generalize matrix multiplication to these higher-order tensors. In particular, a third-order tensor is a three-dimensional array of numbers. The mode i product interprets this tensor as a "stack" of matrices along axis i, and multiplies each matrix in the stack by the same matrix. We can further generalize Freivald's algorithm to verify mode i products between tensors of arbitrary order (with the soundness analysis changing appropriately). We describe this generalization in Appendix B.

6 Video-specific optimizations and a SNARK-friendly video representation

Previous works that build SNARK circuits for editing photos/videos either completely avoid in-circuit compression (as with all papers on image editing that we are aware of) or run (parts of) a traditional compression algorithm in-circuit. While compressing photo/video is obviously useful for reducing storage/networking costs, it does not lead to reductions in prover work/circuit size in a SNARK context. This is because the running time of SNARK provers generally scales in the size of the computation proved. Thus, the work required to produce a proof for circuits that run traditional compression algorithms will be proportional to the size of the uncompressed video.

We propose the concept of SFVR, or SNARK-Friendly Video Representation. An SFVR is a way to represent a video that decreases the work required to process the video in a SNARK circuit. With a suitable SFVR, the work required to produce a proof of video edits is no longer proportional to the uncompressed size of a video. We now suggest a simple SFVR for our system.

Traditional video compression takes advantage of the fact that consecutive frames of video often share a lot of redundancy. Many pixels in a video don't change between subsequent frames or don't change by very much. Furthermore, blocks of pixels may move as groups. Many techniques in traditional video compression are hard to translate to a zk-SNARK context because of their usage of conditional operations and other operations that are hard to translate into circuit form.

However, the basic ideas of video compression can be translated to circuits to obtain an SFVR. Consider a pair of subsequent frames in a video, F_0, F_1. For simplicity, let the frames be 2-dimensional matrices (i.e. grayscale values), though this naturally extends to frames with multiple color channels. Let $F_1 = F_0 + \Delta$. When F_0, F_1 are similar (as is common in many realistic videos), Δ is likely to have low Hamming weight (most pixels don't change between F_0, F_1, so Δ has many 0 entries) and Δ is likely to have low l_1 norm (the pixels that do change don't change by very much). These are common properties of real-world videos, which we validate in Section 9.2.

The types of video editing operations explored in this paper (redactions and convolutions) are compatible with our SFVR. Consider applying a convolution to F_1. We find that

$$A_{\text{left}} F_1 A_{\text{right}} = A_{\text{left}}(F_0 + \Delta) A_{\text{right}} = A_{\text{left}} F_0 A_{\text{right}} + A_{\text{left}} \Delta A_{\text{right}}$$

Looking at the equation, we can see that the edited F_1 can be computed from editing F_0, and applying $A_{\text{left}}, A_{\text{right}}$ to Δ. When Δ has a low l_0 norm, computing $A_{\text{left}} \Delta A_{\text{right}}$ is much cheaper than computing a transformation of F_1 in-circuit, and the zk-SNARK can just output the change in pixels between the edited versions of frame F_0 and F_1. By handling Δ instead of F_1, the editing circuit no longer has to deal with the full uncompressed form of some frames. Redaction operations which replace redacted regions with 0 values are also linear [35], and this technique applies to them too.

Our implementation can also take advantage of Δ having low l_1 norm. We implemented our circuits using the NeutronNova folding scheme [44, 64], which uses elliptic curve-based polynomial commitment schemes (specifically HyperKZG [15] in our implementation). Elliptic curve-based polynomial commitment schemes are faster for values with short bit lengths [9, 12], so our system can take advantage of small l_1 differences between subsequent frames.

To be more concrete about our proposed algorithm, we propose that when signing and compressing a video, as cameras already do, they can sign the "deltas" (or differences) between pairs of close frames. These "deltas" serve as our SFVR. When generating a proof of some video transformation, a prover can either generate a proof that they edited a frame in its uncompressed form, or a proof that they applied some edit to the SFVR (i.e., the difference between two frames). For these SFVR proofs, the verifier verifies both the proof and that the difference between the two corresponding frames in the output video is as expected. To implement an SFVR proof, we take Δ in a sparse matrix representation (e.g., a list of nonzero values and their indices). To compute the edited form of Δ for convolution-style editing operations, we apply Freivald's algorithm with sparse matrix multiplication. This approach is most efficient when the transformed form of Δ is also sparse (or small), which happens for many real-world videos. We describe the exact algorithm implemented and some other practical implementation considerations in Appendix C. For instance, implementing a sparse matrix-vector product requires read/write and read-only memory accesses in a zk-SNARK, and we describe an optimization to decrease the number of more expensive read/write memory accesses.

It is also possible to decrease the l_0, l_1 norms of a video at capture-time to make a video more SNARK-friendly for our SFVR. When experimenting with our test videos, we found that most of the pixels that changed between subsequent frames only changed by one bit. These values can generally be safely changed without affecting the perceptual quality of a video. We propose compressing video to "smooth" it by reducing one bit changes between subsequent frames.

By removing one-bit (or small) changes between subsequent frames of a video, a video compression scheme can force subsequent frames of a video to have lower norm, which means our SFVR consists of even sparser Δ matrices and that the prover is even more efficient.

7 Verifying externally committed data

Recall that verifying video edits requires verifying two independent statements: (i) that the public edited video is the result of applying the specified transformation to an unedited original video (provided as witness data), and (ii) that the unedited original video has a valid signature, such as a C2PA signature. In the previous two sections we looked at how to check (i) efficiently. In this section we return to the problem of checking (ii). Naively, checking that the original unedited video is properly signed requires the SNARK circuit to hash the unedited video and then verify a signature on the hash. Due to the large size of video files, hashing the video makes the SNARK circuit prohibitively large, which motivates the problem of more efficiently verifying that witness data is properly signed.

To formalize the problem, let the video transformation be represented as a relation $\mathcal{R}(x, w) \subseteq X \times W$, where w represents the private, unedited video, and x represents the public edited video along with other public outputs. The relation holds whenever x is the result of applying the claimed transformation to w.

Let us augment the relation $\mathcal{R}$ by adding a requirement that w is properly signed. That is, let us define an augmented instance-witness relation $\mathcal{R}'$ as:

$$\mathcal{R}' := \Big\{ \big((x,\mathsf{pk},\mathsf{com},\sigma),(w,r)\big) \;:\; \mathcal{R}(x,w) = 1 \;\wedge\; V(\mathsf{pk},\mathsf{com},\sigma) = 1 \;\wedge\; \mathsf{com} = \mathsf{Commit}(w,r) \Big\} \tag{3}$$

Here $\mathsf{com} = \mathsf{Commit}(w, r)$ is a hiding and binding commitment to w using randomness r, and $V(\cdot,\cdot,\cdot)$ is a signature verification algorithm. We assume that pk includes a certificate and that V also checks this certificate to ensure that pk is a properly certified public key. This $\mathcal{R}'$ ensures that $\mathcal{R}(x, w)$ holds and that a public commitment to w is properly signed.

VerITAS [24] showed how to prove $\mathcal{R}'$ with a SNARK circuit that is almost the same size as the SNARK circuit for $\mathcal{R}$. Since verifying the signature can be done outside of the SNARK circuit — the verifier can compute $V(\mathsf{pk}, \mathsf{com}, \sigma)$ itself — the expensive step in verifying $\mathcal{R}'$ is verifying that $\mathsf{com} = \mathsf{Commit}(w, r)$ and VerITAS shows how to do that essentially for free. To do so, they set the signed commitment com to be a commitment to a univariate polynomial that encodes w. Now verifying that $\mathsf{com} = \mathsf{Commit}(w, r)$ can be integrated cheaply into the Plonk verifier. While this works well, this approach is limited to the Plonk proof system, and is only feasible for computationally powerful signers (like a generative AI company) because it requires the signer to commit to a high-degree polynomial, which is too costly for a mobile device such as a camera.

In this section we:

- generalize VerITAS's approach for proving $\mathcal{R}'$ to other proof systems,
- eliminate the need for a computationally powerful signer,
- and extend the scheme to support redaction operations, such as cropping or trimming a video. We still use signed polynomial commitments, but no longer require a tight coupling with the underlying proof system.

A core component of these proof techniques is an encoding of the witness w as a polynomial. We will use encodings that are either univariate or multilinear polynomials. We review standard approaches for encoding data as a polynomial and evaluating the resulting polynomial in Appendix D. For example, a common approach to encode a message $m \in \mathbb{F}^n$ as a degree $n-1$ polynomial is to interpolate a polynomial that evaluates to m's entries at a predefined set of n points in the domain of the polynomial.

7.1 Verifying signed polynomial commitments

We begin by generalizing VerITAS's approach for authenticating data beyond the Plonk proof system. Specifically, we work with SNARK proof systems that are the result of applying the Fiat-Shamir transform to an interactive protocol. These types of proof systems allow their circuits to have access to random challenges generated after the prover commits to its witness. Many proof systems support such features, as reviewed in Appendix E.

As before, let $\mathcal{R}(x, w)$ be an instance-witness relation implemented by an arithmetic circuit C. Let g be a polynomial representing the witness w, as in Appendix D, and let σ be a signature on a commitment to g. We aim to build a proof system for the relation $\mathcal{R}'$ from (3) where $\mathsf{Commit}(w, r)$ is replaced with $\mathsf{PCS.Commit}(\mathsf{pp}, g, r)$, for some public parameters pp. We assume that PCS is a hiding and zero-knowledge PCS, as in Appendix A.2.

Let us show how to augment a proof system for $\mathcal{R}(x, w)$ to obtain a proof system Π' for $\mathcal{R}'$. Recall that the circuit C is an implementation of $\mathcal{R}$, that is $C(x, w) = 1$ if and only if $(x, w) \in \mathcal{R}$. The proof system Π' is defined as follows.

- First, the circuit $C(x, w)$ is augmented to a circuit $C'(x, y, z, w)$ with two additional inputs $y, z \in \mathbb{F}$. The circuit C':
  (1) runs $b \leftarrow C(x, w)$,
  (2) evaluates $y' \leftarrow g(z)$ where g is a polynomial that encodes w. This can be done in linear time using polynomial interpolation to evaluate the polynomial g defined by w at the point z.

  The circuit C' accepts if $b = 1$ and $y = y'$. Note that the resulting circuit C' is only marginally larger than C. We will use a proof system for C'.
- An honest prover, given an instance-witness pair for $\mathcal{R}'$, computes the commitment com to w, and uses the Fiat-Shamir hash function applied to the public instance to derive a random challenge $z \in \mathbb{F}$. It then generates a PCS proof π_g that $g(z) = y$. It outputs the final proof as
$$\pi := (\pi_{C'}, \pi_g, y)$$
  where $\pi_{C'}$ is a zk-SNARK proof that $C'(x, y, z, w)$ accepts. That is, $\big((x, y, z), w\big)$ is a valid instance-witness pair for the relation defined by C'. Note that the complete proof sent to the end user who is verifying authenticity of the video is
$$\pi' := (\pi, \mathsf{pk}, \mathsf{com}, \sigma)$$
- The verifier verifies $\big((x, \mathsf{pk}, \mathsf{com}, \sigma), \pi\big)$ as follows: it derives the Fiat-Shamir challenge z, checks the proof $\pi_{C'}$ with respect to

the public instance (x, y, z), checks the proof π_g with respect to the public PCS instance $(\mathsf{pp}, \mathsf{com}, z, y)$, checks the signature σ on com, and accepts if all these checks succeed.

Theorem 7.1. *The proof system Π' for $\mathcal{R}'$ defined above is complete, knowledge sound, and zero-knowledge, assuming $|w|/|\mathbb{F}|$ is negligible, the proof system for C' is a secure zk-SNARK, and the polynomial commitment scheme is hiding and evaluation binding.*

The knowledge soundness of Π' follows from the fact that an extractor for Π' is obtained from an extractor for the proof system for C'. That extractor outputs some witness w so that $C'(x, y, z, w) = 1$. This means that (i) $\mathcal{R}(x, w)$ holds, and (ii) $y = g'(z)$, where g' is a polynomial that encodes w. It remains to argue that com is a commitment to this g' (and is therefore a commitment to w). Since the PCS proof π_g is a valid proof, we know that com is a commitment to a polynomial g that satisfies $y = g(z)$. Therefore, $g'(z) = g(z)$. But since $|w|/|\mathbb{F}|$ is negligible, and z is randomly chosen based on a commitment to g, it follows that $g = g'$ w.h.p., as required.

The zero-knowledge property follows immediately from the zero-knowledge property of the zk-SNARK for C' and the fact that the PCS is hiding.

Our proof system has relatively low costs. The only in-circuit cost for this scheme that scales linearly in the length of w is evaluating $g(z)$ in-circuit. However, this evaluation is concretely cheap, costing 1-3 multiplications per entry of w for common encodings of w as a polynomial.

The technique presented in this section lets us verify the relation $\mathcal{R}'$ from (3) with a small increase to the circuit size due to the polynomial interpolation. In comparison, VerITAS's Plonk-specific approach requires almost no increase in circuit size. In Appendix E.1, we show that the VerITAS approach to authenticating data applies equally well to the Spartan proof system [63] at nearly zero cost to the SNARK circuit size.

We note that our definition of relation (3) is closely related to a Commit-and-Prove SNARK, where we additionally require a signature verification. Proof systems that can show that some values used in their circuits are consistent with a previously computed commitment are called Commit-and-Prove systems. In other contexts, similar methods have also been called "outer commitments" [7], or "precommitted data" [64]. This section's approach primarily differs from previous works in that it does not require a tight coupling with the underlying proof system. We discuss some more related approaches in Appendix E.

7.2 Outsourcing polynomial commitments

The signed polynomial commitment approach of VerITAS [24] is limited to computationally powerful signers, because computing polynomial commitments is concretely expensive. However, we show that polynomial commitments have a desirable property for low power devices like cameras: computing a polynomial commitment can be verifiably outsourced to an untrusted assistant who sees the full unedited video.

Figure 3 shows our outsourcing protocol. The signer (the camera) asks the assistant (a powerful server) to compute a polynomial commitment. After the signer receives a claimed polynomial commitment, the signer asks the assistant to produce an opening proof

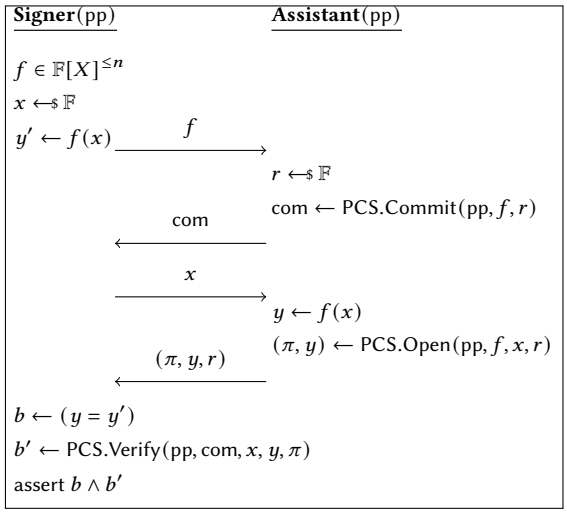

Figure 3: A protocol enabling a weak signer, who has a polynomial $f(X)$, to verifiably outsource computing the PCS commitment to f to an untrusted assistant.

for the polynomial commitment at a random point. The signer locally computes the evaluation and checks that the opening proof is valid and agrees with this local evaluation. Because polynomial evaluation only requires field operations, it is concretely cheap, and thus the signer's work in the outsourcing protocol is substantially smaller than the work required to compute the commitment. This idea is similar to the idea of the *assistant* in Quarks [65], where an untrusted assistant verifies the preprocessing for an R1CS instance by interpolating a polynomial and verifying some opening proofs.

When the signer is a camera, the polynomial f is as large as the video being signed, which leads to potential bandwidth concerns. We address this by having the camera sample the challenge point x and compute $f(x)$ before beginning communication with the server. Assuming the camera securely stores the pair $(x, f(x))$, the camera no longer needs access to the original raw f to complete the protocol. Hence, transmitting f can be done by even physically transferring the camera's memory card to the assistant.

Theorem 7.2. *The protocol in Figure 3 is a secure outsourcing protocol provided that the PCS has evaluation binding and $\deg(f)/|\mathbb{F}|$ is negligible.*

We prove the theorem in Appendix J. At a high level, this is secure when the field $\mathbb{F}$ is sufficiently large because the Schwartz-Zippel Lemma and the security of PCS guarantee that $y' = y$ only holds with negligible probability if com was not honestly computed.

Our approach works the same way for outsourced commitment to a multilinear polynomial. In this case, a multilinear polynomial commitment scheme is used, and the scheme works with $\lceil \log\ n \rceil = \ell$-variate multilinear polynomials. In the multilinear context, outsourcing a commitment to an ℓ-variate polynomial incurs additional soundness loss of $\frac{\ell}{|\mathbb{F}|}$ (rather than $\frac{n}{|\mathbb{F}|}$), and the analysis of the protocol is otherwise the same.

Finally, we briefly mention another variant for computing KZG commitments [41] on the camera, without the help of an assistant. As mentioned previously, we assume that the camera signing a video has a secure element that can store a secret signing key. We propose that the camera's trusted hardware stores the secret exponent τ which is the "toxic waste" generated as part of a KZG trusted setup. This τ would be unique per camera for security. By storing this τ, the camera can compute the polynomial commitment to a polynomial f by computing $f(\tau)$ in trusted hardware, entirely using only field operations. To produce a KZG commitment, the camera only needs to perform one group exponentiation to produce a polynomial commitment to f.

7.3 Efficiently proving redaction

A video is a sequence of frames, say at a rate of thirty frames per second. By video redaction we mean removing a subset of the frames from a video. To prove that redaction was done correctly, one can use Merkle tree-based redactable signatures [39, 67]. This approach is also currently being explored by the C2PA standard [21]. These schemes enable one to efficiently derive a valid signature on a redacted video from a signature on the original video. However, usually the video editor does not only redact a video, but does multiple edits to the video, such as blurring, resizing, and redaction. To prove that blurring and resizing were done correctly on a properly signed video, the polynomial commitment (PCS) techniques used in the previous sections, where one commits to the original unedited video using a PCS decrease the work for a SNARK prover [24, 35]. In Appendix I, we show that if one commits to the original unedited video using a PCS, then redaction can also be efficiently proved using the polynomial commitment. In other words, PCS-based signatures already support redaction, and they can already be used to prove redactions to a video without using another form of traditional redactable signature.

8 Proof system choice and implementation

We used the NeutronNova [44] high-arity folding scheme as implemented in Microsoft's `Spartan2/vega-prover` library [64] as the proof system for our overall implementation. We made this choice because folding schemes are particularly well-suited to generating SNARK proofs of computations with repeated structure like video editing, where the same editing operation is applied to many frames. Furthermore, NeutronNova's prover is relatively easy to parallelize efficiently, as we found in our experiments. While other high-arity folding schemes exist (e.g., Symphony [19]), we chose NeutronNova for its relatively mature implementation.

To scale our implementation of high-arity folding, we made a variety of modifications to the initial NeutronNova implementation. Most notably, we used the approach of Scribe [13] to scale the number of instances folded. Scribe describes an approach that the authors call "Read-Write Streaming", where the state of a SNARK prover is stored on solid-state storage, rather than RAM, allowing the prover to scale to larger problem instances. We adapt their approach to the NeutronNova proof system (though most of our adaptations also apply to uniform instances of the Spartan protocol [9, §7]), which requires some nontrivial modifications to their algorithms. Furthermore, we show that some additional optimizations to Scribe's algorithms are possible when applied to NeutronNova, which allows for fewer streaming passes to be made over the witness data stored in persistent storage, leading to further efficiency gains.

On a hardware level, we show that using high-performance storage, such as a striped storage array, combined with our optimizations to Scribe's algorithms allows for the approach of Scribe to scale to more parallel threads than are used in Scribe's experiments. We also made some lower-level optimizations, such as tuning the library's initial prover implementation to be more parallelized, and switching the underlying polynomial commitment scheme from Hyrax [71] to HyperKZG [15]. These optimizations, while lower-level, lead to meaningful improvements in prover throughput. The details of these optimizations are described in Appendix H. We defer the details since they require more context about the details of NeutronNova and Scribe. As shown by our benchmarks in Section 9.4, these led to a proof system with strong performance characteristics.

Our use of NeutronNova represents a different use of folding schemes from prior work [76] that enables more efficient verification of proofs. Eva uses folding in a traditional manner and performs many recursive folding operations. Each folded instance of a proof in Eva generates part of the final edited video, but only the final proof in a chain of folding instances can output a public value. To deal with this, Eva requires the folded step circuit to evaluate a SNARK-friendly hash of a chunk of the output video in each instance. Eva's SNARK outputs a final hash chain of the output video to cryptographically bind the output of the SNARK to the video a user sees. This also requires the verifier of an Eva proof to apply a SNARK-friendly hash in plain computation. This takes nearly twice as long as watching an input video, even on a 16-core CPU, as can be seen in Eva's evaluations (see Table 5 in [76]). In our implementation of NeutronNova, the folding scheme only applies one folding operation. As implemented in Microsoft's library, each folded circuit instance can have a public output. This means that the output video from the circuits does not need to be hashed with a SNARK-friendly hash by prover or verifier. Instead, the work for the verifier in our scheme is much more hardware-friendly, requiring the verifier to evaluate a hardware-friendly hash of the output video, as well as the multilinear extension(s) of the output video, i.e. the standard work applied to a circuit's public outputs in sumcheck based SNARKS [69]. This is close to the verification procedure of HyperVerITAS. We make one additional change, which is similar to an optimization in VIMz [26]. The approach of HyperVerITAS outputs one field element per byte of the output video. Instead, our circuit "packs" the output bytes of the final video into field elements.[6] This concretely decreases the verifier work by a large factor, decreasing the hashing and multilinear interpolation work.[7]

9 Experiments

We next turn to our performance experiments. We evaluate the performance of our per-frame optimizations, the benefit of our SNARK-friendly video representation, the impact of our signature verification optimizations, and the performance of our optimized

[6] Of course, this requires range checking bytes before doing so.
[7] In our system, this is about 30, since about 30 bytes fit into a BN254 scalar field element.

NeutronNova prover. We conclude with overall system performance and comparison to prior work.

Our large-scale video editing experiments were run on a 64-core server with 512 GB of RAM[8] running an AMD EPYC 7763 CPU from 2021 with access to 5.4 TB of striped solid-state storage. For smaller-scale experiments, we used an M3 MacBook Air with 16 GB of memory. We used the NeutronNova proof system as implemented in the `vega-prover` repository [64] with the `bellpepper` frontend as the base for implementing the circuits for our final experiments, and Scribe's [13] implementation of file-backed vectors/serialization for streaming-based modifications to NeutronNova. For other smaller-scale experiments with authenticating data/implementing editing operations, we used Plonky3 [59], Noir/Barretenberg [10, 11] and arkworks [8]. Our implementation of our core system involved ~2000 lines of modifications to NeutronNova/Scribe's code, ~4000 lines of code specifying new circuits and ~4000 lines of supporting code. We also wrote ~6000 lines of supporting code for the other experiments in this section. The code and data sources for all experiments are available in our repository [30].

9.1 Per-frame benchmarks

First, we benchmark the approach of Section 5.2 for applying a convolution to a 2-dimensional matrix (equivalently, applying some edit to a channel of a video frame). We compare it against a baseline "naive approach," in which a circuit directly implements a convolution by taking linear combinations of values in the matrix. This baseline approach is equivalent to the approaches of works like VerITAS [24] or VIMz [26].

We compare the performance of the two approaches for proving a convolution of a single matrix and show the results in Table 1. We implemented the convolution algorithms in Noir/Barretenberg [10, 11] as an example of a SNARK system that uses Plonkish arithmetization and in Spartan [63] as an example of a SNARK system that uses R1CS arithmetization.

We chose a matrix size of 400×400 for the Noir benchmarks because the Noir compiler could not compile the circuits for the "naive approach" for larger frames and kernel sizes. We chose a larger size of 720×1280 for the Spartan benchmarks because the proof system is faster than Noir, and this made the performance numbers in this table the same order of magnitude. We benchmarked the prover time for both proof systems. To compare these convolution operations to real image transformations, a 3×3 convolution is comparable in complexity to a direct "average blur," which is the type of lower-quality blurring operation implemented in VerITAS/TilesProof and is comparable in complexity to the resizing operation we implement. Our "Gaussian Blur" kernel combines a region of 61×61 pixels (the high definition blur combines many pixels to achieve its quality). For our Freivald's Algorithm variant, the proof generation time is constant with kernel size, though the time to generate a witness and verifier computation (if using the optimization from Section 5.2 when the matrix is public) increases with kernel complexity.

Looking at Table 1, two-sided Freivald's algorithm gives a 1.4-9.5× speedup in prover time over the baseline methods for proving convolutions at example sizes in Noir. The Noir compiler could not compile the circuit for the "naive approach" applied to a 61×61 kernel, but we expect that the speedup factor for such a large kernel would be proportionally larger. This speedup is expected, since addition/linear combination operations are represented as gates in Plonkish arithmetization, and wider linear combinations increase the gate costs of these circuits.

	Naive alg. (Plonkish)	Freivald's alg. (Plonkish)	Naive alg. (R1CS)	Freivald's alg. (R1CS)
Kernel Size	Prover time	Prover Time	Prover time	Prover time
3×3	3.94s	2.81s	2.45s	4.76s
5×5	11.1s	2.81s	5.01s	4.76s
7×7	19.1s	2.81s	8.08s	4.76s
9×9	26.9s	2.81s	14.99s	4.76s

Table 1: Prover performance for proving a convolution of a matrix as kernel size increases.

For our Spartan implementation, the Freivald's algorithm-based approach only becomes faster for kernel sizes larger than 5×5, though it can implement the convolutions for the two larger kernel sizes 1.69-3.14× faster than the naive approach. Interestingly, though the naive approach uses fewer constraints than the Freivald's algorithm approach (since linear combinations do not have gate costs in R1CS), the costs of wide linear combinations eventually outweigh the costs of having fewer constraints in a circuit.

We also compared how closely our implementations of video editing operations matched those in FFmpeg, a standard video editing program, on an example video. To our knowledge, no other works make such a comparison. Our implementation of grayscaling exactly matched FFmpeg's output. Our implementation of image resizing matched FFmpeg's output for 80% of pixels, and the per-pixel deviations between FFmpeg's output and ours had magnitude of at most 1. Our implementation of blurring matched FFmpeg's output for 96.4% of pixels, and the per-pixel deviations between FFmpeg's output and our circuit's were at most 16.

9.2 Video representation benchmarks

In this section, we experiment with our video-specific techniques. We implemented the sparse variant of Freivald's algorithm introduced by Section 6. It costs 12 constraints per nonzero entry of the "delta" matrix. The core component of "sparse Freivald's algorithm" was implementing a sparse matrix-vector product. We describe the optimizations needed to lower the cost of this operation in Section C.

To give some example estimates of performance improvements available from SFVR, we chose our implementation of the Gaussian blurring editing operation. When including the other work performed by our circuits (authentication and rounding), the sparse variant of Freivald's algorithm cost about 2.5× more per changed pixel of the edited image than directly applying Freivald's algorithm, both in terms of gate cost and prover time. Thus, this sparse approach is more efficient than directly implementing Freivald's algorithm for frames where less than 40% of pixels change between subsequent frames.

We experimented with a small set of 13 freely available test videos to estimate the redundancy in real-world videos. We describe

[8]We note that some of our experiments required fewer than 256 GB of RAM.

the test videos used in Appendix F. We analyzed the percentage of pixels which changed between subsequent frames of the video. For frames which changed by less than 40% of their pixels from the previous frame, we recorded the percentage of changed pixels in the frame, and estimated how much cheaper editing the frame with our sparse Freivald's algorithm approach would be. We also experimented with a more realistic estimate, where the system could use circuits to edit frames that change by 5%, 10% and 20% of their pixels or less between subsequent frames. This estimate was more realistic, since our implementation of NeutronNova can only fold identical editing circuits. When integrating SFVR into our prover architecture, we envision having a small number of NeutronNova folding instances for combining editing circuits corresponding to different thresholds of changes between frames for SFVR.

When performing these estimates for our SFVR approach, we found that it can reduce the estimated prover time by 0.2-55.5% for our batch of test videos, with an **average reduction in estimated prover time of** 25.7%. For the more realistic deployment scenario, where there were a small number of fixed circuits for frames where less than 5%, 10% and 20% of pixels change from the previous frame, we found an estimated reduction in prover time of 0.1 − 45.7%, with an **average reduction of 20.4%**. Assuming that changes of ±1 in pixel values are imperceptible to humans, and allowing for a camera to "smooth" these out when compressing this video, the performance of this approach improves to an estimated savings of 32.6% optimistically, and 24.6% realistically. These experiments showed that this approach can reduce the prover work for common videos. Our test videos which did not achieve a very high estimated reduction in prover work were somewhat old YouTube videos shot with a very shaky camera. The videos did not have much redundancy, and better stabilization on modern cameras would potentially improve these scores. These are promising results that show that further developments in SFVR have the potential to help scale zk-SNARKs for video editing operations.

9.3 PCS-based signature benchmarks

Next, we look at the performance of the authentication schemes proposed in Section 7. Figure 4 shows microbenchmarks for the performance of a signer verifiably outsourcing the generation of a polynomial commitment to a more powerful server from Section 7.2. We benchmark evaluating the polynomial representing a message, since this is the dominant component of the signer's running time. As a baseline, we compare against Plonky3's Poseidon [34] implementation, since Poseidon/Poseidon2 are the most common SNARK-friendly hashes and Plonky3's implementation is well optimized. The different types of polynomial interpolation are substantially faster than Poseidon (8.8-30× faster for the largest benchmarked problem sizes), meaning our outsourced signature scheme is not only faster than computing a polynomial commitment, but also faster than traditional SNARK-friendly hashing schemes. Thus, outsourcing the computation of polynomial commitments meaningfully decreases the computation a camera must perform in our system. This is a notable improvement. For example, Eva [76], which uses traditional SNARK-friendly hashing[9] for authentication, quotes a running time of 166 seconds to hash a 2 minute video on a powerful desktop CPU as part of signing [10], longer than the running time of the video. This would likely take much longer on a weak camera processor.

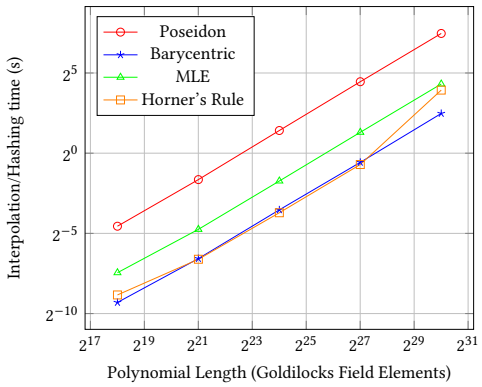

Figure 4: Time to verify outsourced PCS computation. Poseidon is the baseline signing method (i.e., no PCS). Horner's and Barycentric refer to methods to encode a video as a univariate polynomial (in coefficients or evaluations form). MLE refers to encoding the video as a multilinear polynomial.

Approach	Constraints per $\mathbb{F}$ element.	Proving time (s)
Poseidon	132	348.5
Horner's	1	0.696
Barycentric	2	33.67
MLE	3	32.86

Table 2: Comparison of in-circuit methods for evaluating a polynomial derived from a message of 2^{16} $\mathbb{F}$-elements.

Next, we implemented the same type of polynomial evaluation in a SNARK circuit, which is the main computation performed in-circuit by our scheme from Section 7.1 and by the redactable signatures of Section 7.3. We implemented verification for the scheme of Section 7.1 in arkworks [8], specifically a version of Groth16 supporting random in-circuit challenges drawn from the Hekaton project [60]. We show the relative performance of these approaches in Table 2. These interpolation approaches are substantially cheaper inside a circuit than applying Poseidon (at least 10 times faster in prover time). The "Horner's Rule" approach (encoding a message in the coefficients of a polynomial) was the most efficient way of encoding a message as a polynomial for in-circuit evaluation. We note that the per-constraint proving time heavily depended on the structure of the R1CS instance, as well as the constraint counts. Note that the exact constraint costs per $\mathbb{F}$-element depend on the Poseidon parameters used. We also give benchmarks for our redaction schemes based on signed polynomial commitments in Appendix I.3.

[9] They use the Griffin hash, which is comparable to Poseidon2 in plain computation.

[10] They use slightly different approaches for hashing a video as part of signing, and hashing a video for SNARK verification, which is why this number is faster than Eva's verification performance.

Proof System Version	# Instances	Proof Time	Time per Instance
NeutronNova, RAM-based	128 circuits	155.1s	1211ms
NeutronNova, Streaming	128 circuits	181.3s	1416ms
NeutronNova, Streaming	256 circuits	302.8s	1182ms
NeutronNova, Streaming	512 circuits	528.3s	1031ms
NeutronNova, Streaming	1024 circuits	1047s	1022ms

Table 3: Comparing NeutronNova's proving time for different versions of the prover (streaming and non-streaming) and different numbers of folded circuits with $\sim 2^{23}$ constraints.

	Grayscale	Mask	Resize	Gaussian Blur.
Prover Time (s)	4494	4499	8530	11417
Verifier Time (s)	7.99	21.89	13.36	25.97
Peak Storage (TB)	1.711	1.734	2.568	5.136
Proof Size (kB)	651.5	651.5	1635.8	1795.4

Table 4: Performance for proving/verifying various editing operations on a 2 minute, 720p video at 30 fps.

9.4 NeutronNova benchmarks

In Table 3, we benchmark our Read-Write Streaming implementation of the NeutronNova folding scheme. Appendix H contains more numbers and discussion of some lower-level optimizations we made. In this section, we evaluate the overhead of our Read-Write Streaming implementation over using a direct, in-memory prover implementation. As mentioned previously, we used a striped storage array as the backing storage for our implementation. As discussed in Appendix H, using a single NVMe SSD as in the original Scribe paper could not scale to handling 64 concurrent threads for a Read-Write Streaming prover, so we upgraded the Scribe library to support a striped storage system. Table 3 gives the performance of our NeutronNova prover for proving batches of circuits of different sizes, and in different configurations (using RAM or external storage to store its state). The RAM-based prover could not handle batch sizes larger than 128 on our test machine. We also give a time per instance proved to compare the throughput for differently-sized batches. The circuit proven is an example video editing circuit with about 2^{23} constraints. As can be seen, when proving batches of the same size, the streaming NeutronNova implementation is about 17% slower than a RAM-based version, which is comparable to Scribe's overhead for large instance sizes. However, as the batch size is increased, better parallelism and amortization of fixed costs means that larger batch sizes actually achieve a higher prover throughput (17-18% higher at the largest batch size of 1024) over a non-streaming implementation. Somewhat surprisingly, this result shows that this Read-Write streaming approach actually supports *higher* prover throughput at larger batch sizes than a RAM-based implementation, and (in some sense) has no overhead.

9.5 Overall performance

Lastly, we ran benchmarks for our overall system. We used our streaming NeutronNova implementation to show that a video was edited in a valid manner for each of our editing operations in 1 or 2 large folding instances. These circuits included authentication, applying an editing operation,[11] and outputting an edited video. We showed that a 1280×720 pixel, 2-minute video at 30 frames per second was edited correctly with various editing operations. We implemented grayscaling (converting a color video to black and white), masking (blacking out a part of a video), resizing (changing the size of a video) and Gaussian blurring (a higher-quality blurring operation). The last two operations make use of our optimized Freivald's algorithm based approach to video editing. We used the first test video from the benchmarks in Appendix F, cut down to 2 minutes for this purpose.

We give the various resource requirements for these operations in Table 4. The running time of these operations varies from 1 hour and 15 minutes to 3 hours and 10 minutes. In addition to prover and verifier time, we give the peak storage usage for the streaming prover, to show how much persistent storage was used to store prover state.

We compare against the most relevant previous works in Table 5. To compare performance across differently-sized images/videos, we calculate the prover time per pixel, as in Eva [76]. For all prior works which use CPU-based provers, we reran their code on our machine, and chose one of their faster editing operations. For Eva, which uses a GPU, we directly used their performance numbers. We compare the performance of our Gaussian blurring implementation against the performance of some relevant other systems. Gaussian blurring is our slowest operation, but we chose it for its even comparison against Eva. Our Gaussian blurring circuit uses the same number of constraints per pixel as the Eva circuit we compare against (about 21). While our circuits do not handle video compression, the more complicated Gaussian blurring operation uses almost exactly the same number of constraints as Eva's additional video compression and authentication work.

Our system is substantially faster than prior CPU-based editing works. HyperVerITAS [35] has the closest performance to our system. We note that HyperVerITAS does not support arbitrary circuits, and rather builds custom instances of the sumcheck protocol for lattice-based hashes and linear editing operations, rather than supporting arbitrary circuits.

When compared to Eva, our system is about 40% slower, when proving the same number of constraints. Eva uses GPU-based proving. Additionally, Eva has the drawback of using a tiling-based prover, as do multiple of the other prior works. A tiling-based prover is one which splits an input image/video frame into "tiles" or separate blocks, and generates proofs for these separately. Eva only implements operations which edit pixels completely independently. They propose using a vector commitment, like a Merkle Tree to implement editing operations which require accessing the values of pixels in adjacent blocks, for operations like blurring or resizing. However, adding these additional Merkle tree inclusion proofs would make their scheme much slower, since it would substantially increase the in-circuit hashing work required.

Additionally, our system's slowest verification time is about 26 seconds for a 2 minute, 720p video. Eva splits its verification phase into a "hashing" phase and a "SNARK verification" phase, though both are required to verify a video's authenticity. Eva's total verification time for a 2 minute video is 232 seconds. We

[11] This includes rounding pixels, if the relevant edit used fixed point arithmetic

	Format	Prover Time	Max. Dimensions
VIMz[26]†	Image	~ 113µs/px	3840×2160
TilesProof[25]†	Image	~ 64µs/px	6000×4000
VerITAS[24]	Image	~ 36µs/px	6632×4976
HyperVerITAS[35]‡	Image	~ 9.9µs/px	7680×4320
Eva[76]*†	Video	~ 2.6µs/px	1280×720×3600
zk-Cinema	Video	~ 3.4µs/px	1280×720×3600

Table 5: Comparisons against prior work. Max Dimensions refers to the maximum dimensions of media benchmarked in that work. * indicates that the work uses GPU-based proving. †indicates that the work uses tiling-based optimizations. ‡indicates that the work doesn't support arbitrary circuits.

ran our verification using the same number of CPU cores as Eva to make this comparison fair. HyperVerITAS also has relatively long verification times. For instance, their system has verification times of about 100 seconds for the largest benchmarked image sizes of 2^{25} pixels, which are much smaller than the videos we test. TilesProof has verification times of around 3 seconds for a 720×1280 image because it treats the edited video as a public output of multiple Groth16 proofs, which requires MSMs of total size equal to the output image's size. This would also scale unfavorably to a video of 3600 frames. Lastly, when compared to other systems, our Gaussian Blur operation is more complicated than any operation implemented by other systems.

To conclude these comparisons, our work is faster than all relevant prior works which do CPU-based proving, and achieves comparable performance to a GPU-based prior work, especially when factoring in some features our system has which Eva does not.

10 Related work

In Photoproof [53], Naveh and Tromer first introduced the idea of using zk-SNARKs to prove photo edits and created such proofs for 128×128 pixel images. Later work [24–26, 35, 40, 43, 76] improved on this result by creating editing proofs for much larger images through a variety of techniques. Some work used folding techniques to reduce prover time [26, 76], while other work divided large photos into "tiles" and produced separate proofs (in parallel) for each tile. [24, 35] used non-standard hashing functions to reduce the time it takes to prove signature verification. With the exception of [76], all past work exclusively created proofs for photos and not videos. There has also been work on creating editing proofs for audio captured by trusted microphones [4].

Zhang et al. [76] were the first (and so far only) to produce zk-SNARKs for video edits. They optimized encoding and decoding circuits by recognizing that some data is shared between the two algorithms and using this fact to greatly simplify the circuits. They also used optimized folding techniques to reduce prover work for repetitive video editing tasks. Many of the optimizations explored in this work are orthogonal to the optimizations of [76], and could be combined with theirs.

For our specific techniques for video operations (see Section 5.2), we expand on the use of Freivald's algorithm [29], most directly building from [24]. [47] builds optimized sumchecks for proving convolutions that have some resemblance to our approach, concurrent work [35] builds optimized sumchecks for linear operations in photo editing using Freivald's algorithm, and other papers that build optimized sumchecks/GKR variants use approaches that resemble Freivald's algorithm. Our approach primarily differs from these works in that our approach is expressed at the constraint system level, rather than the IOP level, which makes it more extensible. Though this is less efficient, we think it is also valuable to explore approaches for implementing convolutions efficiently within circuits, since these can more easily be combined with arbitrary computations. Compared to [47], our approach to verifying convolutions is more direct and can verify a wider set of operations but requires more verifier work, and [35] does not use the decomposability of kernels, which affects the verification time for their approach. The relation we discuss in Section 7 is closely related to the idea of "Commit-and-prove" SNARKs from works such as [18]. We modify the NeutronNova folding scheme [44], building on streaming techniques from Scribe [13]. Another recent proof system, called Vega [42], also extends NeutronNova using commit-and-prove and other optimizations. However, Vega is primarily designed for proving knowledge of a credential, such as a passport, and not for the large statements that come up in video editing proofs.

11 Conclusions and future work

We present zk-Cinema, a system for generating zk-SNARKs showing that an edited video is authentic and original. We provide improvements like support for more complicated video editing operations, ways of authenticating data more efficiently, schemes for taking advantage of redundancy in videos and an optimized streaming implementation of a high-arity folding scheme. Many of these results may be of independent interest.

Our final system outperforms all prior works which use CPU-based proving. We finally note that our work is aimed at algorithmic and conceptual improvements to video processing and proof systems for zk-SNARKs. Some optimizations and additional features, like GPU acceleration and support for lossy compression, remain. The RAM usage of our system can be reduced via standard techniques like tiling [25, 26, 76] or others [52]. None of the techniques in this paper are incompatible with these future optimizations. The code for this paper is available at [30].

Acknowledgments

This work was supported by NSF, DARPA, and the Simons Foundation. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. We thank Anubhav Baweja, Maurice Shih, Chengru Zhang and Binyi Chen for their helpful discussions about parts of this paper. The authors acknowledge the University of Maryland supercomputing resources (http://hpcc.umd.edu) made available for conducting the research reported in this paper.

References

[1] 2022. Partnership for greater trust in digital photography: Leica and Content Authenticity Initiative. *Leica* (2022). link.
[2] 2022. Sony Unlocks In-Camera Forgery-Proof Technology. *Sony* (2022). link.
[3] 2022. Visuals. *Associated Press* (2022). link.
[4] 2023. ZK Microphone. link.
[5] 2025. C2PA in ChatGPT Images. *OpenAI* (2025). link.
[6] V. Hunter Adams. 2021. Fixed-Point Arithmetic. link.

[7] Sebastian Angel, Eleftherios Ioannidis, Elizabeth Margolin, Srinath Setty, and Jess Woods. 2023. Reef: Fast Succinct Non-Interactive Zero-Knowledge Regex Proofs. USENIX Security 2024. https://eprint.iacr.org/2023/1886

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

[9] Arasu Arun, Srinath Setty, and Justin Thaler. 2023. Jolt: SNARKs for Virtual Machines via Lookups. Cryptology ePrint Archive, Paper 2023/1217. https://eprint.iacr.org/2023/1217

[10] Aztec Network. 2025. barretenberg: An optimized elliptic-curve library and PLONK SNARK prover. https://github.com/AztecProtocol/barretenberg

[11] Aztec Network. 2025. Noir Documentation. https://noir-lang.org/docs/ Version v1.0.0-beta.14.

[12] Suyash Bagad, Quang Dao, Yuval Domb, and Justin Thaler. 2025. Speeding Up Sum-Check Proving. Cryptology ePrint Archive, Paper 2025/1117. https://eprint.iacr.org/2025/1117

[13] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, Karan Newatia, and Steve Wang. 2024. Scribe: Low-memory SNARKs via Read-Write Streaming. USENIX Security 2026. https://eprint.iacr.org/2024/1970

[14] Dan Boneh and Victor Shoup. 2015. *A Graduate Course in Applied Cryptography.* https://toc.cryptobook.us/book.pdf

[15] 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. doi:10.1007/978-3-031-07085-3_15

[16] Vitalik Buterin. 2023. A Quick Barycentric Evaluation Tutorial. HackMD. link.

[17] c2pa [n. d.]. C2PA Technical Specification. link.

[18] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2075–2092. doi:10.1145/3319535.3339820

[19] Binyi Chen. 2025. Symphony: Scalable SNARKs in the Random Oracle Model from Lattice-Based High-Arity Folding. Cryptology ePrint Archive, Paper 2025/1905. https://eprint.iacr.org/2025/1905

[20] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2022. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. Cryptology ePrint Archive, Paper 2022/1355. https://eprint.iacr.org/2022/1355

[21] Coalition for Content Provenance and Authenticity. 2026. *Content Credentials: C2PA Technical Specification.* Specification Version 2.4. Coalition for Content Provenance and Authenticity (C2PA). https://spec.c2pa.org/specifications/specifications/2.4/specs/C2PA_Specification.html

[22] Alistair Coleman. 2022. Ukraine conflict: Further false images shared online. *BBC News* (2022). link.

[23] Alistair Coleman and Shayan Sardarizadeh. 2022. Ukraine conflict: Many misleading images have been shared online. *BBC News* (2022). link.

[24] 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. doi:10.1109/SP61157.2025.00097

[25] 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)*. IEEE Computer Society, Los Alamitos, CA, USA, 4624–4642. doi:10.1109/SP61157.2025.00014

[26] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. 2025. VIMz: Private Proofs of Image Manipulation using Folding-based zkSNARKs. PETS 2025. https://petsymposium.org/popets/2025/popets-2025-0065.pdf

[27] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Paper 2022/1763. https://eprint.iacr.org/2022/1763

[28] France 24 Observers. 2025. *India-Pakistan conflict: AI-generated or edited fake satellite images flood social media.* http://bit.ly/4mPLCci

[29] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time. In *IFIP Congress.* https://api.semanticscholar.org/CorpusID:45405368

[30] Alexander Frolov, Jianfeng Guo, and Xinyi Zhao. 2025. Video Provenance. GitHub. https://github.com/sashafrolov/zk-Cinema

[31] 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

[32] Pascal Getreuer and Paul Mahol. 2011. vf_gblur.c. link.

[33] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML] https://arxiv.org/abs/1412.6572

[34] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. 2023. Poseidon2: A Faster Version of the Poseidon Hash Function. In *AFRICACRYPT 23 (LNCS, Vol. 14064)*, Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne (Eds.). Springer, Cham, 177–203. doi:10.1007/978-3-031-37679-5_8

[35] Garrett Greiner, Toshi Mowery, and Pratik Soni. 2026. HyperVerITAS: Verifying Image Transformations at Scale on Boolean Hypercubes. PETS 2026. https://eprint.iacr.org/2026/641

[36] Stuart Haber, Yasuo Hatano, Yoshinori Honda, William Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. 2008. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security* (Tokyo, Japan) *(ASIACCS '08)*. Association for Computing Machinery, New York, NY, USA, 353–362. doi:10.1145/1368310.1368362

[37] Ulrich Haböck. 2022. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530. https://eprint.iacr.org/2022/1530

[38] David G. Harris. 2008. Simultaneous field divisions: an extension of Montgomery's trick. Cryptology ePrint Archive, Paper 2008/199. https://eprint.iacr.org/2008/199

[39] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. 2002. Homomorphic Signature Schemes. In *CT-RSA 2002 (LNCS, Vol. 2271)*, Bart Preneel (Ed.). Springer, Berlin, Heidelberg, 244–262. doi:10.1007/3-540-45760-7_17

[40] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation. arXiv 2211.04775.

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

[42] Darya Kaviani and Srinath Setty. 2026. Vega: Low-Latency Zero-Knowledge Proofs over Existing Credentials. In *IEEE Symposium on Security and Privacy*. IEEE, 1951–1969. doi:10.1109/SP63933.2026.00182

[43] Hankyung Ko, Ingeun Lee, Seunghwa Lee, Jihye Kim, and Hyunok Oh. 2021. Efficient Verifiable Image Redacting based on zk-SNARKs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (Virtual Event, Hong Kong) *(ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 213–226. doi:10.1145/3433210.3453110

[44] Abhiram Kothapalli and Srinath Setty. 2024. NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Paper 2024/1606. https://eprint.iacr.org/2024/1606

[45] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2022. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *CRYPTO 2022, Part IV (LNCS, Vol. 13510)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Cham, 359–388. doi:10.1007/978-3-031-15985-5_13

[46] Abhiram Kothapalli and Srinath T. V. Setty. 2024. HyperNova: Recursive Arguments for Customizable Constraint Systems. In *CRYPTO 2024, Part X (LNCS, Vol. 14929)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 345–379. doi:10.1007/978-3-031-68403-6_11

[47] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. CCS 2021. https://eprint.iacr.org/2021/673

[48] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083 [stat.ML] https://arxiv.org/abs/1706.06083

[49] Paul Mahol. 2016. vf_avgblur.c. link.

[50] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S. Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. 2020. Think Global, Act Local: Gossip and Client Audits in Verifiable Data Structures. arXiv:2011.04551 [cs.CR] https://arxiv.org/abs/2011.04551

[51] Matt Murphy, Olga Robinson, and Shayan Sardarizadeh. 2025. Misleading posts obtaining millions of views on X. *BBC* (2025). link.

[52] Vineet Nair, Justin Thaler, and Michael Zhu. 2025. Proving CPU Executions in Small Space. Cryptology ePrint Archive, Paper 2025/611. https://eprint.iacr.org/2025/611

[53] 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. doi:10.1109/SP.2016.23

[54] Michael Niedermayer. 2001. swscale.c. link.

[55] Alex Ozdemir, Evan Laufer, and Dan Boneh. 2024. Volatile and Persistent Memory for zkSNARKs via Algebraic Interactive Proofs. Cryptology ePrint Archive, Paper 2024/979. https://eprint.iacr.org/2024/979

[56] Seongbin Park, Alexander Vilesov, Jinghui Zhang, Hossein Khalili, Yuan Tian, Achuta Kadambi, and Nader Sehatbakhsh. [n. d.]. Chimera: Creating Digitally Signed Fake Photos by Fooling Image Recapture and Deepfake Detectors. *Usenix Security 2025* ([n. d.]).

[57] Valerie Pavilonis. 2022. Fact check: Images show Mosul in 2017, Kyiv one day after Russian invasion began. *USA Today* (2022). link.

[58] Wahid Pessarlay. 2024. Nikon, Canon, Sony eye tamper-resistant digital signatures to combat deepfakes. *CoinGeek* (2024). link.

[59] Polygon Technology. 2023. Plonky3. link.

[60] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. 2024. Hekaton: Horizontally-Scalable zkSNARKs Via Proof Aggregation. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 929–940. doi:10.1145/3658644.3690282

[61] Todd Rowland and Eric W. Weisstein. 2025. Tensor. MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/Tensor.html
[62] Scroll Technologies. 2025. Ceno: Accelerated Zero-knowledge Virtual Machine by Non-uniform Prover Based on GKR Protocol. https://github.com/scroll-tech/ceno Accessed: 2025-05-15.
[63] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Cham, 704–737. doi:10.1007/978-3-030-56877-1_25
[64] Srinath Setty. 2020. Spartan: High-speed zkSNARKs without trusted setup (Spartan2). https://github.com/microsoft/Spartan2. GitHub repository, accessed 2025-11-19.
[65] Srinath Setty and Jonathan Lee. 2020. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Paper 2020/1275. https://eprint.iacr.org/2020/1275
[66] Srinath Setty, Justin Thaler, and Riad Wahby. 2023. Unlocking the lookup singularity with Lasso. Cryptology ePrint Archive, Paper 2023/1216. https://eprint.iacr.org/2023/1216
[67] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. 2002. Content Extraction Signatures. In *ICISC 01 (LNCS, Vol. 2288)*, Kwangjo Kim (Ed.). Springer, Berlin, Heidelberg, 285–304. doi:10.1007/3-540-45861-1_22
[68] James Sturgis. 2014. public test videos. https://gist.github.com/jsturgis/3b19447b304616f18657. GitHub Gist, accessed 2025-11-19.
[69] Justin Thaler. 2022. *Proofs, Arguments, and Zero-Knowledge.* https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf
[70] ThinkParQ GmbH. 2026. BeeGFS – The Leading Parallel Cluster File System. https://www.beegfs.io. Accessed: 2026-07-24.
[71] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. 2017. Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2017/1132. https://eprint.iacr.org/2017/1132
[72] Anna P. Y. Woo, Alex Ozdemir, Chad Sharp, Thomas Pornin, and Paul Grubbs. 2025. Efficient Proofs of Possession for Legacy Signatures. Cryptology ePrint Archive, Paper 2025/538. doi:10.1109/SP61157.2025.00080
[73] Yibin Yang and David Heath. 2023. Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM. Cryptology ePrint Archive, Paper 2023/1115. https://eprint.iacr.org/2023/1115
[74] Ian T. Young and Lucas J. van Vliet. 1995. Recursive implementation of the Gaussian filter. *Signal Process.* 44, 2 (June 1995), 139–151. doi:10.1016/0165-1684(95)00020-E
[75] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2024. BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes. In *CRYPTO 2024, Part X (LNCS, Vol. 14929)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 138–169. doi:10.1007/978-3-031-68403-6_5
[76] Chengru Zhang, Xiao Yang, David Oswald, Mark Ryan, and Philipp Jovanovic. 2025. Eva: Efficient Privacy-Preserving Proof of Authenticity for Lossily Encoded Videos. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 4643–4662. doi:10.1109/SP61157.2025.00237

A Deferred Security Definitions

A.1 Digital Signatures

From [14], for a signature scheme to be existentially unforgeable under chosen message attack, the probability that adversary $\mathcal{A}$ wins the following game must be negligible:

- The challenger generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}$ and sends pk to the adversary
- $\mathcal{A}$ can issue Q signature queries to the challenger. Query i for $i = 1, \ldots Q$ is a message m_i. In response, the challenger sends $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m_i)$ to $\mathcal{A}$
- $\mathcal{A}$ outputs a forgery pair (m^*, σ^*)

The adversary wins if $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 1$ and if $m^* \notin \{m_1, \ldots m_Q\}$.

A.2 Polynomial Commitments

A polynomial commitment scheme must be correct and evaluation binding and can additionally be hiding. We repeat the following definitions from [24]:

- **Correctness**: for all $\lambda, \mu, d \in \mathbb{N}$, all $\vec{x} \in \mathbb{F}^\mu$, and all $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\leq d}$, the following probability is 1:

$$\Pr\left[\begin{array}{l|l:l} \mathsf{Vf}(\mathsf{pp}, \mathsf{com}, & & \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mu, d) \\ \vec{x}, y, \pi) = 1 & : & r \leftarrow\!\!\$\ \mathcal{R}_{\mathrm{C}} \\ & & \mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, f, r) \\ & & (\pi, y) \leftarrow \mathsf{Open}(\mathsf{pp}, f, \vec{x}, r) \end{array}\right]$$

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

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

A polynomial commitment scheme can be

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

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

- **Zero-knowledge** if it is hiding and for every PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that for all $f \in \mathbb{F}[X_1, \ldots, X_\mu]^{\leq d}$, all $r \in \mathcal{R}_{\mathrm{C}}$, all $\vec{x} \in \mathbb{F}^\mu$, and $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, f, r)$, $(\pi, y) \leftarrow \mathsf{Open}(\mathsf{pp}, f, \vec{x}, r)$ is a zk-SNARK (see Section A.3) for the following relation:

$$\left\{\begin{array}{c|c:c} (\mathsf{pp}, \pi, \vec{x}, y, \mathsf{com}); & & \mathsf{com} = \mathsf{Commit}(\mathsf{pp}, f, r) \\ (f, r) & : & \wedge\ f(\vec{x}) = y \end{array}\right\}$$

A.3 zk-SNARKs

A zk-SNARK must be complete, knowledge sound, zero-knowledge, non-interactive, and succinct. We repeat the following definitions from [24]:

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

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

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

$$\Pr\left[\begin{array}{l|l:l} (x, w) \in \mathcal{R} & & \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda) \\ & : & (x, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp}) \\ & & w \leftarrow\!\!\$\ \mathcal{E}^{\mathcal{A}_1(\mathsf{state})}(\mathsf{pp}, x) \end{array}\right] \geq$$

$$\Pr\left[\begin{array}{l|l:l} \mathsf{Vf}(\mathsf{pp}, x, \pi) = 1 & & \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda) \\ & : & (x, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}_0(\mathsf{pp}) \\ & & \pi \leftarrow\!\!\$\ \mathcal{A}_1(\mathsf{state}) \end{array}\right] - \epsilon$$

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

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

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

A.4 Folding Schemes

Folding schemes [44] allow a prover to reduce the work of proving multiple instances of the same relation to the work of proving a single instance of the relation. Let

$$(u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, \{w_i\}_{i\in[m]}), \mathcal{V}(\mathsf{vk})\rangle(\{u_i\}_{i\in[m]})$$

denote the verifier's output instance u and the prover's output witness w from the interaction of $\mathcal{P}$ and $\mathcal{V}$ on witnesses $\{w_i\}_{i\in[m]}$, prover key pk, verifier key vk, and instances $\{u_i\}_{i\in[m]}$.

A folding scheme must be perfectly complete and knowledge-sound and may be high arity. We repeat the following definitions from [44].

- Perfect Completeness: a folding scheme is perfectly complete if for all PPT adversaries $\mathcal{A}$ and all natural numbers n,

$$\Pr\left[(\mathsf{pp}, s, u, w) \in \mathcal{R} : \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, n) \\ (s, \{(u_i, w_i)\}_{i\in[m]}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \forall i \in [m],\ (\mathsf{pp}, s, u_i, w_i) \in \mathcal{R} \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s) \\ (u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, \{w_i\}_{i\in[m]}), \mathcal{V}(\mathsf{vk})\rangle(\{u_i\}_{i\in[m]}) \end{array} \right] = 1$$

- Knowledge Soundness: a folding scheme is knowledge-sound if for any expected polynomial-time adversary $\mathcal{P}^*$ and any natural number n, there is an expected polynomial-time extractor $\mathcal{E}$ s.t.

$$\Pr\left[\forall i \in [m],\ (\mathsf{pp}, s, u_i, w_i) \in \mathcal{R} : \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(\lambda, n) \\ (s, \{u_i\}_{i\in[m]}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho) \\ \{w_i\}_{i\in[m]} \leftarrow \mathcal{E}(\mathsf{pp}, \rho) \end{array} \right] \geq$$
$$\Pr\left[(\mathsf{pp}, s, u, w) \in \mathcal{R} : \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(\lambda, n) \\ (s, \{u_i\}_{i\in[m]}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \rho) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s) \\ (u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, \rho), \mathcal{V}(\mathsf{vk})\rangle(\{u_i\}_{i\in[m]}) \end{array} \right] - \mathsf{negl}(\lambda)$$

  where ρ denotes arbitrary randomness for $\mathcal{P}^*$.
- High-arity: a folding scheme is high-arity when $m > 2$.

B Applying Freivald's algorithm to nth order tensors

This section continues the discussion of Freivald's Algorithm from Section 5.3. At a high level, the main observation of this section is that the analysis of Freivald's algorithm largely works the same if the relevant matrices are replaced with nth order tensors (i.e. n dimensional arrays of numbers), and matrix multiplication is replaced with the ith mode product (i.e. interpreting the tensor as a collection of vectors along some dimension i, and multiplying each vector by the same matrix). This intuitively makes sense, since the ith mode product is a generalization of matrix multiplication. As before, we are viewing tensors in the "machine learning" sense, where they are just multidimensional arrays of numbers.

Consider a product of an nth order tensor, $A \in \mathbb{F}^{m_1 \times m_2 \ldots \times m_n}$ with a matrix $B \in \mathbb{F}^{m_i \times m_i'}$. We are looking to verify the product $A \times_i B$, with claimed evaluation C. C is also an nth order tensor of the same dimensions as A, except with its ith dimension replaced with m_i'. For our variant of Freivald's algorithm, we sample a vector r with m_i' entries. To verify the product $A \times_i B = C$, we compute $A \times_i (Br)$ and $C \times_i r$, and verify that the resulting tensors are equal.

Generalizing the analysis of the previous section, C has $m_1 \times \ldots \times m_i' \times m_n$ entries, and this protocol has a soundness error of $\frac{m_1 \ldots m_i' \ldots m_n}{|\mathbb{F}|}$. Considering its efficiency, computing the two tensor products requires computing 2 mode i products between an order n tensor and a vector, as well as a matrix-vector product. This is strictly more efficient than computing a mode i product between an order n tensor and a matrix.

This basic approach generalizes to verifying a convolution as explained in Section 5.2. This involves verifying both a left and right mode i product, sampling 2 sets of random vectors of the correct dimensions, and precomputing the products of the left and right matrices with the randomly sampled vectors.

C Extended SFVR discussion

The main algorithmic component of our implementation of SNARK-friendly video representation from Section 6 was a sparse matrix-vector product to compute the quantity $A_{\text{left}} \Delta A_{\text{right}}$ for a sparse matrix Δ. We chose to implement the following simple algorithm for sparse matrix-vector products:

```
1 : // A simple sparse matrix-vector product between the
2 : // m × n A with d nonzero entries and vector v. A is
3 : // represented as a list of triples of value and coordinates.
4 : SparseMatrixVectorProduct(A : F^{d×3}, v : F^n) :
5 :   output ← [0, ..., 0] ∈ F^m
6 :   for(val, i, j) ∈ A :
7 :     output[i] = output[i] + val * v[j]
8 :   return output
```

We implemented relatively standard algorithms for both read/write memory, and read-only memory. We used LogUp [37] for read-only memory, and the approach of Yang and Heath [73] for read/write RAM. Read-only RAM operations are cheaper than Read/Write RAM operations under these protocols. In implementing the sparse matrix-vector product algorithm directly, processing each entry of

A requires one Read-only memory operation, and two Read/Write RAM operations (to read the value of v[j] and read/write output[i] respectively).

We implement a simple optimization to reduce these costs. Specifically, we change A's representation to contain "strips" of values. A's representation contains groups of B entries, each with the same i coordinate, but potentially different j coordinates. Now, when computing an output, SparseMatrixVectorProduct only needs to perform two read/write operations for every B entries of A (to read and write the current/former value in coordinate i of output). At the cost of a small overhead for unused entries of these "strips" (when B does not divide the number of nonzero entries in a given row), this decreases the cost of RAM operations for these sparse products.

We note that SFVR does leak which frames in a given video do not change by much from prior frames. This is a potential drawback of the approach. We believe that this small leakage is acceptable for most applications. Another potential issue to mention is rounding. Many of the operations we implement are not quite linear, because video editing algorithms must round the outputs of a linear operation to values between 0 and 255. There are a few potential ways of dealing with this. Some prior systems more or less ignore in-circuit rounding, like VerITAS [24]. Otherwise, adding 2 rounded values together can only impact the resulting value by at most 1. In most cases, this is acceptable, and a SNARK could show that these errors are small. These deviations would be edit/video dependent, rather than arbitrarily controlled by an adversary.

D Representing videos as polynomials

In our approaches for verifying a signed polynomial commitment and outsourcing the computation of a polynomial commitment, the signer and verifier must both evaluate a large polynomial at a random point, where the polynomial encodes a message (some witness data). We briefly review three natural ways of representing a message as a polynomial that can be evaluated efficiently, both in regular computation and in arithmetic circuits. In this context, we call a polynomial representing a message an *extension polynomial*. We chose this term since the extension polynomial of a message in the multilinear case is the message's multilinear extension.

In our multivariate schemes, we represent a message as a polynomial by using its multilinear extension (padding the message with zeros to make its length a power of two, if needed). To compute the multilinear extension of a vector $m \in \mathbb{F}^{2^\ell}$, we represent it as a function $m' : \{0,1\}^\ell \to \mathbb{F}$, where $m'(i)$ returns the $int(i)$-th entry of m for $i \in \{0,1\}^\ell$. The multilinear extension of m can then be evaluated using the standard interpolation formula:

$$f(x) = \sum_{b \in \{0,1\}^\ell} m'(b) \cdot \prod_{i|b_i=1} x_i \cdot \prod_{i|b_i=0} (1 - x_i) \tag{4}$$

As explained in Chapter 5 of [69], this formula can be evaluated in three multiplications per $b \in \{0,1\}^\ell$.

In our univariate schemes, we can use two standard approaches to represent a message as a polynomial. Consider a message $m \in \mathbb{F}^n$. Using the "coefficient basis" [55], we can encode the m in the coefficients of the polynomial $f(x) = m_0 + m_1 x + m_2 x^2 + \ldots + m_{n-1} x^{n-1}$. Using Horner's rule, f can be rewritten as

$$f(x) = m_0 + x(m_1 + x(m_2 + x(m_3 + x(m_4 + \ldots))))$$

This implies a streaming algorithm to evaluate $f(x)$ which requires one field multiplication and addition per coefficient.

A more standard approach to represent a message $m \in \mathbb{F}^n$ as a univariate polynomial is in the "evaluation basis". Specifically, we encode m in the evaluations of a polynomial f, which satisfies $f(\omega^i) = m_i$ for all $i \in [n]$, where ω is a root of unity of order 2^ℓ such that $2^\ell \geq n$. This polynomial f can be evaluated in linear time using the barycentric formula [16] for polynomial evaluation:

$$f(x) = \frac{x^n - 1}{n} \times \sum_i \frac{y_i * \omega^i}{x - \omega^i} \tag{5}$$

This can be computed in $O(2^\ell)$ field operations per interpolated value. Outside of a SNARK circuit, this evaluation can be computed quickly using Montgomery's trick [38] for inverting a batch of values efficiently.

To conclude, these three extension formulas give efficient ways — in a SNARK circuit and on a regular processor — to evaluate a polynomial that encodes a message $m \in \mathbb{F}^{2^\ell}$ or $m \in \mathbb{F}^n$. We can use all of these methods to encode m as a polynomial and verify a signature on it, or verify that a polynomial commitment to m was correctly computed. We benchmark these different approaches in Section 9.3.

E Outer commitments

In this section we first review the concept of "outer commitments". The term was first defined in [7], though the technique appears under various names in earlier works. An outer commitment [7] is a commitment that is in some sense native to a zk-SNARK's underlying proof system. This is closely related to the idea of "Commit-and-Prove SNARKs" [18]. Crucially, the commitment is public, and showing that wire values in the circuit are consistent with the commitment does not need to be expressed in constraints. These approaches can also be used to access pseudorandom values in-circuit without additional in-circuit work. In the context of this paper, we can use outer commitments to authenticate data without additional in-circuit work by having the verifier check a signature (in our case from the camera) on the outer commitment (in our case to the video data). For all proof systems supporting outer commitments that we are aware of, an outer commitment is a Pedersen commitment or polynomial commitment.

We list some examples of outer commitments from previous work:

- The "Commit-and-Prove" technique of LegoSNARK [18] can be thought of as an outer commitment, since it can be used to input data into a Groth16 proof by using a Pedersen commitment to the relevant data.
- Nova [45] allows for outer commitments, as explained in Reef [7]. The authors use Pedersen commitments to input data into a Nova proof and authenticate it.
- Using copy constraints and a modified Plonk permutation argument, a variant of Plonk can support an outer commitment, as described in VerITAS [24].

- Spartan can be modified to support outer commitments, which we explain in Section E.1.

These approaches are particularly useful for computations represented by "wide but shallow" circuits that need to verify signatures. We call circuits wide but shallow if they have a large witness and/or public input, but perform a small amount of computation per element of the witness/public input. In this case, the costs of verifying a signature on the data outweigh the cost of the relevant computation. Video/photo edits are often wide but shallow, as are some machine learning and database applications.

The Spartan and Plonk approaches to outer commitments are the most flexible outer commitment schemes, while the Reef/Groth16 approaches do not support the signature schemes we describe later because they do not involve polynomial commitments.

E.1 Outer commitments for Spartan

This section assumes familiarity with the internals of the Spartan proof system [63]. As mentioned previously, the VerITAS system [24] provides a way to verify Relation (3) from Section 7, at zero cost to the circuit (i.e. no in-circuit polynomial evaluation), while the approach suggested in Section 7.1 shows how to verify Relation (3) with a small increase to the circuit size. The VerITAS technique is specific to Plonk [31], while the method in Section 7.1 is generic. Here, we show how to adapt the VerITAS technique to the Spartan proof system [63].

Spartan is a sumcheck-based SNARK for R1CS, where different parts of the R1CS relation $(A \cdot z) \circ (B \cdot z) = C \cdot z$ are expressed as multilinear polynomials. A, B, C are matrices expressing constraints, $\circ$ is the Hadamard (i.e. entrywise) product, and $z = (io, 1, w)$ is the "witness vector" made by concatenating the public inputs/outputs io, the scalar 1 and the private witness w. Let $n = |z|$ and $\ell = \log n$. Without loss of generality, $n/2 = |io| + 1 = |w|$. When we write a tilde above a vector, like $\widetilde{(io, 1)}$, it denotes the multilinear extension of that vector. The equation for the multilinear extension of the R1CS trace vector $z = (io, 1, w)$ in Spartan is defined as:

$$\widetilde{Z}(\vec{x}) \;=\; (1 - x_0)\widetilde{(io, 1)}(x_1, \ldots, x_{\ell-1}) + x_0\widetilde{w}(x_1, \ldots, x_{\ell-1}) \quad (6)$$

This creates the multilinear extension of $(io, 1, w)$ by combining the multilinear extensions of $(io, 1)$ and w using a new variable x_0[12].

We observe that Spartan can be easily extended to use the data encoded by a precomputed polynomial commitment as part of the witness in a Spartan proof. Let v be a vector, and let $\widetilde{v}$ be its multilinear extension. Without loss of generality, let $|v| = |w| = n/4$ and $n/2 = |v| + |w| = |io| + 1$.

Suppose that the verifier has a polynomial commitment to $\widetilde{v}$. We observe that a small modification to Spartan lets the verifier be assured that the R1CS "trace vector" z is $z = (io, 1, v, w)$. That is, the verifier is assured that both io (which the verifier has explicitly) and v (for which the verifier only has a commitment) are correctly embedded in z. To do so, we write the multilinear extension of z as:

$$\begin{aligned}\widetilde{Z}(\vec{x}) = {} & (1 - x_0)\widetilde{(io, 1)}(x_1, \ldots, x_{\ell-1}) \\ & + x_0(1 - x_1)\widetilde{v}(x_2, \ldots, x_{\ell-1}) \\ & + x_0 x_1 \widetilde{w}(x_2, \ldots, x_{\ell-1})\end{aligned} \quad (7)$$

When evaluating $\widetilde{Z}(\vec{x})$ at a random point at the end of the protocol, the verifier uses openings of both $\widetilde{v}$ and $\widetilde{w}$. The rest of the analysis of Spartan works the same way with this modification (except for modifying the circuit, and how the prover/verifier evaluate $\widetilde{Z}(\vec{x})$ accordingly). This observation is similar to a technique used in a concurrent work [72], which uses a similar idea to create a variant of Spartan that supports "multi-stage" circuits.

[12]This is the "Merge PIOP" in Hyperplonk's [20] terminology.

F Test videos for SFVR

In Table 6, we show a sample of test videos from a small dataset of test videos [68] that we used for this paper, and report numbers from our experiments estimating the impact of SNARK friendly video representation on the prover time. The dataset consists of freely available online videos, which include 3D animations made to demonstrate the Blender software, ads for Google's Chromecast product and vlogs about cars. In general, videos with a shaky camera had much less redundancy that could be used by SFVR. These videos were filmed in 2010, and better camera stabilization algorithms would likely improve the performance of SFVR for these videos.

G Fixed-point arithmetic considerations

Some of our image transformations require arithmetic with fractional components, which is difficult to implement in arithmetic circuits over finite fields. We use fixed-point arithmetic for these operations.

To recall fixed-point arithmetic [6] (we use a binary variant), set some precision n. To represent a value $x \in \mathbb{R}$ with n bits of fractional precision as an integer value, we represent it as $\lfloor x * 2^n \rfloor$. To multiply 2 fixed-point numbers x, y, we compute $\lfloor x * y * 2^{-n} \rfloor$. To implement this in a finite field of prime order, we interpret multiplication and addition as finite field operations and interpret multiplication by 2^{-n} and taking the floor as operations over the natural representation of a field element as an integer.

It takes some care to mix fixed-point arithmetic with Freivald's Algorithm correctly, while also achieving the highest possible precision. In particular, when multiplying matrices ABC, we cannot perform the "reduction" step of multiplying by 2^{-n} and taking the floor in between multiplying AB and $(AB)C$, since this breaks the linearity needed for Freivald's Algorithm. Additionally, if working in a small field, we must ensure that the finite-field arithmetic does not "wrap around," (by setting the precision based on the number of bits in the finite field) so that we accurately emulate floating-point operations.

H NeutronNova implementation and optimizations

As previously mentioned, we adapted and optimized the NeutronNova proof system from Microsoft's `vega-prover` (formerly known as `Spartan2`) repository [64] for our experiments.[13] In this appendix, we describe our optimizations in greater depth. This section assumes familiarity with the NeutronNova folding scheme [44], Scribe and Read-Write Streaming SNARKs [13], and general usage of the sumcheck protocol [69].

[13]The NeutronNova implementation has subsequently been removed from the repository after it was renamed. The base version of NeutronNova we built off of can be found here.

Table 6: Description of test videos.

Name		Description	Theoretical/Realistic SFVR Improvement	Theoretical/Realistic SFVR Improvement with Smoothing
BigBuckBunny.mp4		3D Animation	15.8%/9.3%	25.7%/17.0%
ElephantsDream.mp4		3D Animation	14.4%/6.8%	25.2%/12.7%
ForBiggerBlazes.mp4		Advertisement	50.2%/42.5%	54.7%/47.0%
ForBiggerEscapes.mp4		Advertisement	55.6%/45.8%	63.6%/51.9%
ForBiggerFun.mp4		Advertisement	12.0%/9.2%	18.0%/11.4%
ForBiggerJoyrides.mp4		Advertisement	50.6%/42.4%	56.5%/46.4%
ForBiggerMeltdowns.mp4		Advertisement	53.4%/45.8%	57.5%/49.2%
Sintel.mp4		3D Animation	6.6%/3.7%	19.4%/7.3%
SubaruOutbackOnStreetAndDirt.mp4		Vlog	23.0%/19.2%	26.6%/23.8%
TearsOfSteel.mp4		Live-action movie	24.4%/17.3%	36.5%/23.5%
VolkswagenGTIReview.mp4		Vlog	17.0%/12.4%	23.0%/17.9%
WeAreGoingOnBullrun.mp4		Vlog	0.2%/0.1%	3.2%/0.2%
WhatCarCanYouGetForAGrand.mp4		Vlog	11.6%/10.5%	13.5%/11.0%

H.1 General Optimizations

We made a number of optimizations to the initial NeutronNova proof system implementation, which were more general and not specific to the Scribe approach. First, when we started working on this project, many important operations in `vega-prover`'s implementation of NeutronNova were not parallelized, leading to suboptimal performance. We worked with the maintainers to debug and reintroduce this parallelism back into their code. In this pass of optimizations, a part of the loop for the main NeutronNova sumcheck, witness generation and folding witnesses were parallelized. On our 64 core machine, this led to a 1.8× improvement in prover throughput over the prior version.[14]

Next, `vega-prover`'s implementation of NeutronNova implements the "small-value sumcheck" approach of [12]. This version of the sumcheck algorithm is specialized for the case when most or all of the evaluations of the polynomial that the sumcheck protocol is applied to have small bitlengths. For instance, the polynomials generated from applying Spartan or NeutronNova to a circuit computing SHA-256 hashes would mostly evaluate to 0 or 1, since these circuits mostly use Boolean witness values. Almost all of the circuits we implemented did not benefit from these optimizations, since in our setting most witness values are nearly random field elements (arising from computations like intermediate values for LogUp, or from applying Freivald's algorithm). By removing these checks, many inner loops of the NeutronNova code were made simpler. Surprisingly, these optimizations alone led to an approximately 13% improvement in prover throughput.

The initial implementation of the NeutronNova prover used many short parallel sections separated by sequential sections. This is a common undesirable programming pattern in parallel computing. There is some cost in waiting for all threads in a parallel section to finish, so it is more desirable to have fewer, longer parallel sections. We merged witness generation, polynomial commitment generation for the separate instances and some other minor steps into one parallel section. This led to a further approximately 5% performance improvement.

[14] We got these numbers by running NeutronNova's prove for proving a batch of 64 circuits corresponding to a simulated video editing operation, and dividing by 64 to get a time per frame edited.

To summarize, by implementing these optimizations, we improved the performance of the NeutronNova prover for a test circuit by 2.13×.[15]

We later implemented a Read-Write streaming version of the NeutronNova prover, which allowed us to prove much higher arity folding operations than before. There are some fixed costs to NeutronNova proving, which are amortized when folding more instances together. By enabling higher arity folding, our streaming implementation was more efficient per instance proved because of the ability to amortize fixed costs over more folded instances.

Most of the optimizations we discuss in this section are fairly standard parallel computing optimizations. However, we think it is notable that they led to a more than 2× performance improvement, even in a relatively mature library like `vega-prover`. Many of these issues are less noticeable on machines with 8 or 16 threads, but became more pressing performance bottlenecks on machines with 64/128 cores, like the ones we used in this work. This may provide evidence that other SNARK libraries may also be amenable to these types of optimizations.

On top of this previous set of optimizations, we also switched the polynomial commitment scheme used by NeutronNova from Hyrax [71] to HyperKZG [15]. The initial implementation of NeutronNova used the Hyrax polynomial commitment scheme, largely because of the availability of the PCS. Switching out Hyrax for HyperKZG gave another 40-50% improvement in prover throughput for large circuits. This was because the main step in computing a polynomial commitment with HyperKZG is one large multiscalar multiplication (MSM), while Hyrax's PCS involves approximately $\sqrt{n}$ MSMs of size $\sim \sqrt{n}$. Consequently, Hyrax benefits much less from the speedups associated with Pippenger's algorithm.

H.2 Streaming NeutronNova optimizations

As summarized previously, the core idea of Read-Write Streaming SNARKs is to store intermediate proof states on external storage instead of RAM. This allows provers to handle larger SNARK instances. To maximize the efficiency of this approach, the prover must read or write to the states stored on disk purely sequentially (i.e. in a streaming way). Scribe, the prior work which introduces the idea of Read-Write Streaming SNARKs, implements their experiments using the HyperPlonk [20] proof system. While this is a reasonable choice of proof system, there are some interesting considerations in applying their approach to NeutronNova. At a high level, the repeated structure of the operations proved by NeutronNova enables some optimizations which are not available to an implementation of a monolithic proof system using Scribe's approach. Some of these optimizations trade memory usage (Scribe uses only 750 MB of prover RAM) for faster performance.

First of all, the initial phase of NeutronNova (before folding instances together) involves generating witnesses for each NeutronNova instance, and then computing polynomial commitments to each witness, as with any folding scheme. For witness generation/commitment generation, Scribe implements streaming versions of polynomial commitment schemes and a witness generation algorithm. Making these steps use streaming is often unnecessary in the case of NeutronNova. In a single instance of a high-arity folding scheme, the polynomial commitments to each folded instance's witness can be computed mostly independently. Furthermore, as long as the size of the circuit for each folded instance is small enough to fit in RAM, then this polynomial commitment computation and witness generation can be done in a non-streaming manner, and the only disk operation required in this initial computation is for writing the generated witness vector to disk. This optimization removes one reading pass over the entire initial witness to the proof, at the cost of some increase in RAM usage.

Next, we turn to our optimizations to the folding part of NeutronNova, which involves proving an instance of the sumcheck protocol whose size scales in the combined size of each folded instance. We describe our optimizations in terms of Spartan for uniform circuits. This has been studied before in systems like Jolt [9, 12]. The core sumcheck instance proved in `vega-prover`'s implementation of NeutronNova is slightly more complicated, but uniform Spartan captures the main ideas of our system.

In Spartan, the goal is to prove the following instance of the sumcheck protocol:

$$0 = q(\mathbf{S}) = \sum_{y \in \{0,1\}^{\log m}} \widetilde{eq}(\mathbf{S}, y) \cdot \left(\left(\sum_{x \in \{0,1\}^{\log n}} \widetilde{A}(y,x) \cdot \widetilde{\mathbf{u}}(x) \right) \circ \left(\sum_{x \in \{0,1\}^{\log n}} \widetilde{B}(y,x) \cdot \widetilde{\mathbf{u}}(x) \right) - \sum_{x \in \{0,1\}^{\log n}} \widetilde{C}(y,x) \cdot \widetilde{\mathbf{u}}(x) \right)$$

This equation is taken from Section 4 of [52]. In this equation, S is a random challenge, $\widetilde{eq}$ is the standard multilinear equality polynomial, $\widetilde{A}, \widetilde{B}, \widetilde{C}$ are the multilinear extensions of matrices representing an R1CS instance, and $\widetilde{\mathbf{u}}$ is the multilinear extension of the R1CS witness vector. In Spartan, a random challenge $\mathbf{S}$ is derived, and the sumcheck protocol is invoked to check that this sum evaluates to 0 with challenge $\mathbf{S}$.

Consider the structure of A, B, C when they represent a uniform circuit (i.e. a circuit which contains parallel, unrelated instances of the same exact circuit). In this case, (as discussed in [12] and in the Jolt repository[16]), A, B, C can be rearranged to have a block diagonal structure. Call the number of repeated circuit instances k. Without loss of generality, let k be a power of 2, and assume that the dimensions of the blocks in the block diagonal structure of A, B, C are also powers of 2.

When applying the approach of NeutronNova to the main sumcheck for Spartan (as the implementation in the `vega-prover` repository does), the resulting sumcheck instance only differs slightly from the Spartan sumcheck applied to a uniform constraint system. The main differences are that it is only run for the first $\log k$ rounds, a homomorphic polynomial commitment scheme which commits to the per-instance witnesses separately is used (rather than a monolithic one for the entire witness), and the equality polynomial is replaced with a related polynomial called τ.[17]

[15] These benchmarks from during development were obtained using a version of the prover which used the Hyrax polynomial commitment scheme, rather than the HyperKZG one which we ultimately used.

[16] https://github.com/a16z/jolt/issues/347

[17] $\tau(\vec{x})$ evaluates to $\tau^{\text{bin}(x)}$ for points $\vec{x}$ on the Boolean hypercube. The details of this are described in the NeutronNova paper. Please see the original NeutronNova implementation for details of this sumcheck instance: https://github.com/microsoft/vega-prover/blob/0d4f1409e8f30536b8b25ed3f81bc446ed717e61/src/neutronnova_zk.rs

First of all, this block diagonal structure allows for an intermediate step of Spartan proving to be parallelized. The vectors $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$ (which are often computed in implementations of Spartan), can be computed in parallel and in small space because their segments for each parallel instance contribute to independent segments of the matrix vector products. This provides a different way of achieving parallel witness generation which we mentioned previously.

This block diagonal structure and precomputation of $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$ also motivates our last two optimizations. The existing implementations of Spartan and NeutronNova in the `vega-prover` repository precompute the matrix vector products $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$. Then, when implementing the main Spartan sumcheck, they perform this sumcheck on the vectors $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$ rather than reevaluating the inner sums which range over $x \in \{0,1\}^{\log n}$.

We made one last optimization to the way these matrix-vector products are stored which would not typically make sense in a context where there is abundant RAM. Specifically, we combine the fact that Spartan for uniform constraint systems (when k is a power of 2) allows for segments of $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$ to be computed without storing the whole witness vector in memory, and that the operation of "binding" sumcheck variables is linear (i.e. that $(1-r_i)Au_j + r_iAu_{j+1} = A((1-r_i)u_j + r_iu_{j+1})$). We only store the vectors u during the invocation of the sumcheck protocol, and recompute the products $A\mathbf{u}, B\mathbf{u}, C\mathbf{u}$ for each segment when generating proofs.

Considering the costs of this optimization, it leads to a decrease in persistent storage used (approximately a 3× decrease when A, B, C are close to square), which means our implementation uses persistent storage much more efficiently. This also leads to a performance improvement, since in most cases, the performance improvement from lower usage of disk bandwidth outweighs a small number of additional field operations to recompute these products.

Lastly, combining these last 2 optimizations, we make one more optimization which allows us to avoid approximately one read pass over the combined Spartan/NeutronNova witnesses. In Scribe's implementation of the sumcheck protocol, and in general, there are two main steps. There is a "folding" step, in which the vector of evaluations of the relevant multilinear polynomials is folded in half using a random challenge, and a step in which the coefficients of a "round polynomial" are computed. In Scribe, these two steps are performed with separate streaming passes. However, again, the block diagonal/segmented structure of the constraint system allows us to "fold" a segment of $\mathbf{u}$, and then persist this segment in RAM to use it to compute this segment's contribution to the round polynomial. This again removes a reading pass from the execution.

In summary, by assuming that the individual instances in NeutronNova are small enough, our implementation of NeutronNova can avoid performing reads to disk for certain operations and perform certain per-instance work in the same way an in-memory prover would, removing multiple read passes from our streaming implementation and decreasing the amount of required disk operations.

H.3 Scaling Read-Write Streaming

The original Scribe paper [13] evaluates their implementation using an NVMe SSD and at most 12 threads used by the prover. This is relatively low compared to the 64 threads we used in our benchmarks. In their benchmarks, Scribe's streaming Hyperplonk implementation achieves a 10-20% overhead over a regular RAM-based Hyperplonk implementation. When optimizing our implementation of NeutronNova, we realized that Scribe's approach cannot be scaled using only NVMe SSDs. When scaling past 32 threads, the overhead of the approach increased substantially, from around 10%, to 100% or more, because the code was exceeding the read/write capacity of the SSD interface. On our test machine, we had access to a fast networked storage array, which used striped storage. It specifically used the BeeGFS file system [70], with storage striped across 4 storage nodes. This striped storage array had better throughput than any one NVMe SSD. Using the high-performance file system this provided, we were able to achieve overhead numbers comparable to Scribe's while using 64 or even 128 concurrent prover threads. These types of high-performance file systems are common in high-performance computing, AI, and data center applications. We think this result is notable because high-performance file systems/high-performance storage are a "fancy" type of hardware which has not yet been applied to scaling zk-SNARKs. However, this result shows that they have some utility in building faster/larger zk-SNARKs.

H.4 Conclusion

In conclusion, by (i) applying some software engineering/parallel computing practices, (ii) implementing NeutronNova-specific optimizations to the techniques of Scribe, and (iii) using hardware that is better-suited to the problem, we were able to build an optimized folding-based prover with fast performance and the ability to scale to very large instance sizes.

I Proofs of Redaction

I.1 Efficiently proving redaction with univariate polynomials

In this section we show how to enhance the VerITAS PCS signature verification method to directly support proving a redaction operation, like trimming a video. Given a signed polynomial commitment to a message $m = (m_0, \ldots, m_{n-1}) \in \mathbb{F}^n$, a prover can prove that a subinterval of m is equal to some public m'. Instead of proving this with generic SNARK techniques, our proofs consist of at most three PCS evaluation proofs, which are cheaper to compute than a SNARK. We describe a proof system using a univariate PCS here and defer the description of a proof system using a multilinear PCS to Appendix I.2.1.

At a high level, we describe how to prove that some public message m' is a subinterval of a signed univariate polynomial commitment to a message $m = (m_0, \ldots, m_{n-1})$. More formally, let C be a univariate PCS with public parameters pp, let Sig be a digital signature scheme with public key pk, and denote the univariate extension of m as $\mathsf{uext}(m) = m_0 + m_1 X + \ldots + m_{n-1}X^{n-1}$. We describe a SNARK for the following relation $\mathcal{R}^n_{\mathsf{urpcs}}$:[18]

$$\left\{ \begin{array}{cll} (\mathsf{pp}, \mathsf{pk}, m', a, & & m \in \mathbb{F}^n \ \wedge \ m[a:b] = m' \\ \quad b, \mathsf{com}, \sigma); & : & \wedge\ \mathsf{com} = \mathsf{C.Commit}(\mathsf{pp}, \mathsf{uext}(m), r) \\ \quad (m, r) & & \wedge\ \mathsf{Sig.Vf}(\mathsf{pk}, \sigma, \mathsf{com}) = 1 \end{array} \right\}$$

[18] Multiple proofs can be combined when the structure of the redacted interval is more complicated.

Say we have some $m = (m_0, \ldots, m_{n-1})$. For any $a, b \in [n]$ where $b > a$, and claimed public subinterval $m' = (m'_0, \ldots, m'_{b-a-1})$, we can define:

$$L(X) := m_0 + \ldots + m_{a-1}X^{a-1}$$
$$C(X) := m'_0 + m'_1X + \ldots + m'_{b-a-1}X^{b-a-1}$$
$$H(X) := m_b + m_{b+1}X + \ldots m_{n-1}X^{n-b-1}$$

Clearly $m' = m[a : b]$ if and only if:

$$\mathsf{uext}(m) = L(X) + X^a C(X) + X^b H(X) \tag{8}$$

By the Schwartz-Zippel Lemma, if the prover shows that (8) holds at a random point α, then with very high probability $m[a : b] = m'$. Because m' is public, the verifier can compute $C(\alpha)$ itself. If the prover is given a commitment to $\mathsf{uext}(m)$, then to allow the verifier to check that (8) holds at a random α, the prover only needs to commit to $L(X)$ and $H(X)$ and provide evaluation proofs for $[\mathsf{uext}(m)](\alpha)$, $L(\alpha)$, and $H(\alpha)$. To complete the proof, the prover must also prove that $L(X)$ and $H(X)$ have degrees $< a$ and $< n - b$ respectively.

We now describe how to modify the FRI polynomial commitment scheme so that evaluation proofs prove both polynomial openings and degree bounds. A standard FRI evaluation proof guarantees that a committed polynomial f has degree less than 2^ℓ for some ℓ. To allow a prover to use this property to show that a committed polynomial has degree $< n$ for an arbitrary $n < 2^\ell$, we use a "saturating" approach. When the prover is given some polynomial f, instead of committing to f, it commits to $g = f \cdot X^{2^\ell - n}$. When the verifier receives a valid opening proof that $g(x) = y$, it computes $f(x) = y/x^{2^\ell - n}$ and is convinced that f has degree $< n$. This incurs a small soundness loss compared to the soundness of the standard FRI evaluation proof. Other PCS evaluation proofs provide similar degree bounds, so this technique can be adapted to other polynomial commitment schemes, though this scheme is written in the context of FRI.[19]

For the rest of this section assume that C is the FRI PCS and that C.Commit and C.Open take an additional parameter $D = 2^\ell$ that enforces that the committed polynomial has degree $< D$. We also assume C has an additional deterministic algorithm $\mathsf{Recover}(x, y, D, n)$ that returns y/x^{D-n}. The verifier uses Recover to obtain the evaluation for a polynomial whose commitment was computed with our saturating approach (i.e., Recover is only necessary when $n < D$).

Figure 5 describes our prover and verifier algorithms for $\mathcal{R}^n_{\mathsf{urpcs}}$. We state the protocol as an interactive protocol made non-interactive via the Fiat-Shamir heuristic. com is a commitment to $\mathsf{uext}(m)$ computed with randomness r and degree bound n (we assume n is a power of 2). The prover computes commitments to $L(X)$ and $H(X)$ using the saturating approach from above, so the verifier must run C.Recover to obtain evaluations of $L(X)$ and $H(X)$.

The verifier verifies the signature and evaluation proofs and then rederives the Fiat-Shamir random challenge α. To obtain $\alpha_L = L(\alpha)$ and $\alpha_H = H(\alpha)$, the verifier runs C.Recover. As discussed earlier, this convinces the verifier that com_L is a commitment to a polynomial of degree $< a$ and that com_H is a commitment to a polynomial of degree $< n - b$. The verifier computes $\alpha_C = C(\alpha)$ itself. Because com is computed with degree bound n, the verifier simply sets $[\mathsf{uext}(m)](\alpha) = y_m$ and then checks that Equation 8 holds with respect to $[\mathsf{uext}(m)](\alpha) = y_m$, $\alpha_L = L(\alpha)$, $\alpha_C = C(\alpha)$, and $\alpha_H = H(\alpha)$. The verifier accepts if all checks pass and rejects otherwise.

[19] For instance, in KZG commitments, the polynomial's degree is bound to be less than the length of the trusted setup.

Prover for $\mathcal{R}^n_{\mathsf{urpcs}}(\mathsf{pk}, \mathsf{pp}, m \in \mathbb{F}^n, a \in [n], b \in [n], r \in \mathbb{F}, \mathsf{com}, \sigma)$

$f \leftarrow \mathsf{uext}(m)$
$(D_L, D_H) \leftarrow (2^{\lceil \log a \rceil}, 2^{\lceil \log(n-b) \rceil})$
$L(X) \leftarrow \mathsf{uext}(m_0, ..., m_{a-1}) \cdot X^{D_L - a}$,
$H(X) \leftarrow \mathsf{uext}(m_b, ..., m_{n-1}) \cdot X^{D_H - (n-b)}$
$r_L, r_H \leftarrow\$ \mathbb{F}$
$\mathsf{com}_L \leftarrow \mathsf{C.Commit}(\mathsf{pp}, L(X), r_L, D_L)$
$\mathsf{com}_H \leftarrow \mathsf{C.Commit}(\mathsf{pp}, H(X), r_H, D_H)$
$\alpha \leftarrow \mathsf{H}(a, b, m[a : b], \mathsf{com}, \mathsf{com}_L, \mathsf{com}_H, \sigma)$
$(\pi_m, y_m) \leftarrow \mathsf{C.Open}(\mathsf{pp}, f, \alpha, r, n)$
$(\pi_L, y_L) \leftarrow \mathsf{C.Open}(\mathsf{pp}, L(X), \alpha, r_L, D_L)$
$(\pi_H, y_H) \leftarrow \mathsf{C.Open}(\mathsf{pp}, H(X), \alpha, r_H, D_H)$
return $(\sigma, (\mathsf{com}, \pi_m, y_m), (\mathsf{com}_L, \pi_L, y_L), (\mathsf{com}_H, \pi_H, y_H))$

Verifier for $\mathcal{R}^n_{\mathsf{urpcs}}(\mathsf{pp}, \mathsf{pk}, a, b, m' \in \mathbb{F}^{b-a}, \pi)$

$(\sigma, (\mathsf{com}, \pi_m, y_m), (\mathsf{com}_L, \pi_L, y_L), (\mathsf{com}_H, \pi_H, y_H)) \leftarrow \pi$
Check $\mathsf{Sig.Vf}(\mathsf{pk}, \mathsf{com}, \sigma) = 1$
$\alpha \leftarrow \mathsf{H}(a, b, m', \mathsf{com}, \mathsf{com}_L, \mathsf{com}_H, \sigma)$
Check $\mathsf{C.Vf}(\mathsf{pp}, \mathsf{com}, \alpha, y_m, \pi_m) = 1$
Check $\mathsf{C.Vf}(\mathsf{pp}, \mathsf{com}_L, \alpha, y_L, \pi_L) = 1$
Check $\mathsf{C.Vf}(\mathsf{pp}, \mathsf{com}_H, \alpha, y_H, \pi_H) = 1$
$(D_L, D_H) \leftarrow (2^{\lceil \log a \rceil}, 2^{\lceil \log(n-b) \rceil})$
$\alpha_L = \mathsf{C.Recover}(\alpha, y_L, D_L, a)$
$\alpha_H = \mathsf{C.Recover}(\alpha, y_H, D_H, n - b)$
$\alpha_C = m'_0 + ... + m'_{b-a-1}\alpha^{b-a-1}$
Check $y_m = \alpha_L + \alpha^a \alpha_C + \alpha^b \alpha_H$.
return 1 if all checks passed and 0 otherwise.

Figure 5: Prover and Verifier for $\mathcal{R}^n_{\mathsf{urpcs}}$

Intuitively, this is secure because the FRI evaluation proof ensures that the polynomial committed to by com is a degree-n polynomial. Furthermore, each coefficient in this degree-n polynomial must come from one of $L(X)$, $C(X)$, or $H(X)$ because of the degree bound checks on com_L and com_H. We use the Schwartz-Zippel lemma to show that the degree bound checks are secure and that a dishonest prover can only pass this verifier check with small probability. We rigorously analyze our scheme in Appendix J. Note that σ and com can be reused multiple times to prove the membership of multiple subintervals for the same message.

The redaction transformation described in this section can also be implemented using traditional redactable signatures [36], which let one construct a signature on a subset of a vector from a signature on the entire vector. However, the proof-based approach in this section is better suited to video editing in SNARKs because the resulting proof can be combined with proofs for other transformations applied to the video, resulting in a single succinct proof.

We discuss variants of this scheme for multilinear polynomials in Appendix I.2.1, and show how our proposed schemes connect to lookup arguments like Caulk/cq [27] or Lasso [66].

I.2 Redactable signatures from multilinear polynomials

We now describe analogs of the redactable signature algorithms in Section 7.3 that work with multilinear polynomials. They serve the same function, but use a different set of techniques for multilinear polynomials.

I.2.1 Proving Redaction with Multilinear Polynomials. In this section, we describe two analogs of the approach of Section 7.3 which make use of multilinear polynomial commitments. First, we show a scheme for proving membership of a single interval of elements in a piece of signed data. Specifically, given a signed multilinear polynomial commitment to a message $m = (m_0, \ldots, m_{n-1})$ that can be divided into n/L blocks, we can prove that some public message m' is the i-th block of m. This block-based approach could be useful for showing that one frame is included in a video as part of distributing the generation of SNARK proofs of editing each frame across many different workers.

Let C be a multilinear PCS with public parameters pp, and let Sig be a digital signature scheme with public key pk. For ease of exposition, we assume that n is a power of 2 and that L perfectly divides n; if this is not the case, we can easily pad m so that the scheme still works. We implement this padding in our experiments. We describe a SNARK for the following relation $\mathcal{R}^{n,L}_{\text{mlrpcs}}$ where $i \in [n/L]$:

$$\left\{ \begin{array}{cc} (\mathsf{pp}, \mathsf{pk}, m', & m \in \mathbb{F}^n \ \wedge \ m' = m[iL : (i+1)L] \\ i, \mathsf{com}, \sigma); & : \ \wedge \ \mathsf{com} = \mathsf{C.Commit}(\mathsf{pp}, \mathsf{MLE}(m), r) \\ (m, r) & \wedge \ \mathsf{Sig.Vf}(\mathsf{pk}, \sigma, \mathsf{com}) = 1 \end{array} \right\}$$

Let $\mathsf{bin}(i)$ denote the binary decomposition of integer i. Our proof relies on the following fact about multilinear extensions. Let n be a power of 2, let $m \in \mathbb{F}^n$, and let L perfectly divide n (i.e., L is also a power of 2). For any $i \in [n/L]$, let $m' = m[i(L) : (i+1)L]$ be the claimed public subinterval. Then for any $i \in [n/L]$ and any $\alpha \in \mathbb{F}^{\log L}$, the following holds:

$$\mathsf{MLE}(m)(\mathsf{bin}(i)||\alpha) = \mathsf{MLE}(m')(\alpha) \tag{9}$$

By the Schwartz-Zippel Lemma, if the prover can show that Equation 9 holds at a random α, then with very high probability, $m' = m[iL : (i+1)L]$. Because m' is public, the verifier can compute $\mathsf{MLE}(m')(\alpha)$ itself. Given a commitment to $\mathsf{MLE}(m)$, the prover can provide an evaluation proof for $\mathsf{MLE}(m)(\mathsf{bin}(i)||\alpha)$, which enables the verifier to check Equation 9.

Figure 6 shows our prover and verifier algorithms for $\mathcal{R}^{n,L}_{\text{mlrpcs}}$. We describe this as a protocol made non-interactive using the Fiat-Shamir heuristic. We formally analyze the protocol in Appendix J.

The signature σ and commitment com can be reused, and the approach can be extended to show the membership of an interval of I blocks in a message with $O(\log\ I)$ opening proofs by an argument similar to [50]. However, proving membership of a set of indices in a message represented by a signed multilinear polynomial commitment of an irregular length can be done more efficiently via a sum-check approach using standard techniques for sumcheck protocols. We outline one possible approach later in this section.

Prover for $\mathcal{R}^{n,L}_{\text{mlrpcs}}(\mathsf{pk}, \mathsf{pp}, m \in \mathbb{F}^n, L, i \in [n/L], r \in \mathbb{F}, \mathsf{com}, \sigma)$

1: $m' \leftarrow m[iL : (i+1)L]$
2: $\alpha \leftarrow \mathcal{H}(m', \mathsf{com}, i, \sigma) \in F^{\log L}$
3: $\beta \leftarrow \mathsf{bin}(i)||\alpha$
4: $(\pi, y) \leftarrow \mathsf{C.Open}(\mathsf{pp}, \mathsf{MLE}(m), \beta, r)$
5: **return** $(\sigma, (\mathsf{com}, \pi, y))$

Verifier for $\mathcal{R}^{n,L}_{\text{mlrpcs}}(\mathsf{pp}, \mathsf{pk}, m' \in \mathbb{F}^L, i \in [n/L], \mathsf{com}, \sigma, \pi, y)$

1: Check $\mathsf{Sig.Vf}(\mathsf{pk}, \mathsf{com}, \sigma) = 1$
2: $\alpha : F^{\lceil \log L \rceil} \leftarrow \mathcal{H}(m', \mathsf{com}, i, \sigma)$
3: $y' \leftarrow [\mathsf{MLE}(m')](\alpha)$
4: $\beta \leftarrow \mathsf{bin}(i)||\alpha$
5: Check $\mathsf{C.Vf}(\mathsf{pp}, \mathsf{com}, \beta, y, \pi) = 1$
6: Check $y' = y$
7: **return** 1 if all checks passed and 0 otherwise.

Figure 6: Multilinear redaction proving

I.2.2 Multilinear redactable signatures via the Zero Check PIOP. Showing the membership of an interval of values in a message represented by a signed polynomial commitment to a multilinear polynomial, especially when the redacted interval does not fit into a block nicely, can be done more directly via a sumcheck approach. More formally, if we let C be a multilinear PCS with public parameters pp and we let Sig be a digital signature scheme with public key pk, we describe a proof system for the following relation $\mathcal{R}^{n}_{\text{mlrsc}}$

$$\left\{ \begin{array}{cc} (\mathsf{pp}, \mathsf{pk}, m', a, & m \in \mathbb{F}^n \ \wedge \ m[a : b] = m' \\ b, \mathsf{com}, \sigma); & : \ \wedge \ \mathsf{com} = \mathsf{C.Commit}(\mathsf{pp}, \mathsf{MLE}(m), r) \\ (m, r) & \wedge \ \mathsf{Sig.Vf}(\mathsf{pk}, \sigma, \mathsf{com}) = 1 \end{array} \right\}$$

Let $f(X) = \mathsf{MLE}(m)$. Define the *mask polynomial* $v(X)$ to be a multilinear polynomial such that over the Boolean hypercube, $v(\mathsf{bin}(i)) = 1$ for $i \in [a, b]$, and $v(\mathsf{bin}(i)) = 0$ for all $i \in [n] \setminus [a, b]$. Let the *revealed message* m' be the public subinterval, and let $g(X) = \mathsf{MLE}(0^a \ || \ m' \ || \ 0^{n-b})$.

Clearly $m' = m[a : b]$ if and only if f and g agree on the points on the Boolean hypercube that correspond to $[a, b]$. In other words, $m' = m[a : b]$ if and only if $g(\mathsf{bin}(i)) - f(\mathsf{bin}(i)) = 0$ for all $i \in [a, b]$.

Let $h(X) = f(X)v(X) - g(X)$. Because $v(\mathsf{bin}(i)) = 0$ for all $i \in [n] \setminus [a, b]$, then $h(\mathsf{bin}(i)) = 0$ for all $i \in [n]$ if and only if $m' = m[a : b]$. We can prove $h(X)$ is zero everywhere on the hypercube using a *zerocheck* from [20], which requires a single sumcheck invocation. In more detail, to show that $m[a : b]$ agrees with m' at the relevant indices, the prover runs a sumcheck on $(f(X)v(X) - g(X)) \cdot \mathrm{eq}(X, r)$ for a random r provided as challenge by the verifier. At the end of this sumcheck protocol, the verifier needs to check the evaluation of $h(X)$ at a random point α. To enable this, the prover sends an evaluation proof for $f(\alpha)$, and the verifier computes $v(\alpha)$ and $g(\alpha)$ itself to obtain $h(\alpha)$.

Unlike the univariate scheme, this scheme can prove more complicated patterns of redaction than just intervals $[a, b]$ (any pattern of 0/1 that can be encoded in $v(x)$). We give a more formal security analysis in Appendix J.

I.2.3 A connection to lookup arguments. The Caulk/cq line of lookup arguments [27] and the nlookup/Lasso [46, 66] line of lookup arguments are formalized as techniques for showing that a set of values in a circuit are consistent with the evaluations of a committed-to polynomial over some basis. Caulk, cq and related techniques are used to show that a set of values in a circuit consists of the evaluations of a polynomial committed to by a KZG commitment over some basis. nlookup/Lasso are techniques for showing that a set of values in a circuit are all the evaluations of a committed-to or cheap-to-evaluate multilinear polynomial at points on the Boolean hypercube.

In this lens, the techniques in this section can be viewed as variants of polynomial commitment-based lookup arguments where batching is particularly efficient when the looked-up values are all in one block, or as an optimization for the case of "sequential reads" from a lookup table.

I.3 Performance of PCS-based redaction schemes

We also benchmarked our redaction schemes from Section 7.3. To implement the univariate redaction scheme, we used Plonky3's `TwoAdicFriPcs` polynomial commitment scheme. To implement the multilinear redaction scheme, we used Ceno's [62] implementation of Basefold [75] with Reed-Solomon codes in Plonky3's ecosystem. These were chosen because they represent comparable coding-theoretic polynomial commitment schemes for univariate and multilinear polynomials, and both implementations were reasonably well optimized. We used an example (uncompressed) video which was 720×1280 pixels and 10 seconds long. We tested two scenarios: showing that a single frame is contained in a video (useful for distributing proving across multiple proofs) and showing that a range of frames is contained in the signed video (useful for representing a trimmed video). The benchmark results for the first scenario are in Table 7, and the results for the second in Table 8. For the second table, we benchmarked revealing a relatively large interval of the test video. The row labeled "Underlying PCS time" shows the time for the underlying polynomial commitment scheme used to commit to a polynomial of relevant size, since this is the dominant cost for generating an authenticated commitment for both signature schemes.

To summarize the performance differences between the two schemes, the univariate scheme involves committing to data when proving redaction, while the multilinear scheme does not. However, the amount of work required for the univariate scheme decreases as the revealed interval's size increases, while the work increases for our multilinear scheme. However, the performance of these two schemes is largely dictated by the performance of the underlying polynomial commitment scheme used.

When comparing against a baseline of using Merkle Trees with Poseidon, it takes about 10.5 seconds to build a Poseidon2 Merkle Tree (using Plonky3's implementation) over the same reference video, and 3 ms per inclusion proof. While generating and verifying Merkle Proofs is much faster than the times for our redactable signature scheme, they are much less circuit-friendly, as shown by Table 2. Since generating a SNARK proof of a Poseidon2 hash of a reasonably large revealed segment of a video would most likely involve committing to a trace polynomial of much larger degree than the length of the original video, our two redactable signature schemes are better when revealing any reasonably large fragment of data authenticated by a polynomial commitment in a SNARK context.

Step	Univariate	Multilinear
Generating Signed Commitment (s)	303.0	16.7
Proving Redaction (s)	630.2	21.29
Verifying Redaction (s)	0.064	0.025
Underlying PCS time (s)	297.5	16.69

Table 7: Univariate vs. Multivariate redaction schemes for proving membership of one frame in 10s, 720p uncompressed reference video.

Step	Univariate	Multilinear
Generating Signed Commitment (s)	301.7	16.17
Proving Redaction (s)	390.66	24.7
Verifying Redaction (s)	0.646	1.25
Underlying PCS time (s)	296.1	16.16

Table 8: Univariate vs. Multivariate redaction schemes for proving membership of range of frames in 10s, 720p uncompressed reference video.

J Deferred proofs of security

Theorem J.1. *The protocol of Figure 3 is a public-coin succinct interactive argument of knowledge for the instance-witness relation* $\mathcal{R}_{\mathsf{PCS}} = \{((\mathsf{pp}, f, \mathsf{com})\ ;\ r)\ :\ \mathsf{com} = \mathsf{PCS.commit}(\mathsf{pp}, f, r)\}$ *when* $|\mathbb{F}|$ *is sufficiently large.*

Proof: Translating a similar proof from [65] into a univariate context, this protocol is complete, succinct and public-coin because the polynomial commitment scheme is assumed to have these properties. Since the polynomial commitment scheme is assumed to be extractable, there is a polynomial f' such that com is a commitment to $f'(x)$. By the binding property of a polynomial commitment scheme $f'(x) = y = y' = f(x)$. Since x is randomly chosen, the Schwartz-Zippel Lemma applies, and $f'(x) = f(x)$ except for a soundness error of $\frac{n}{|\mathbb{F}|}$.

Next, we prove the security of the univariate redactable signature scheme by interpreting it as an interactive protocol (i.e. replacing steps where we hash values with interactions where the signer sends the relevant commitments, and the verifier sends a public random challenge).

Theorem J.2. *We claim that the prover and verifier algorithms in Figure 5, when interpreted as an interactive protocol, are a succinct public-coin argument of knowledge for the instance-witness relation* $\mathcal{R}^n_{\mathsf{urpcs}}$*, i.e. that the signature authenticates that the revealed message is a subinterval of the actual message.*

The succinctness and public-coin properties follow from the properties of the underlying polynomial commitment scheme and by inspection on the messages sent.

This protocol is also complete. An honest signer sends polynomial commitments to $\mathsf{uext}(m_0, \ldots, m_{n-1})$, $\mathsf{uext}(m_0, \ldots, m_{a-1}) \cdot$

X^{D_L-a}, $\mathsf{uext}(m_b, \ldots, m_{n-1}) \cdot X^{D_H-(n-b)}$, and then opens these values at the random challenge point α. Call these openings y_m, y_L and y_H respectively. Clearly $\mathsf{uext}(m_0, \ldots, m_{a-1})(\alpha) = \frac{y_L}{\alpha^{D_L-a}}$ and $\mathsf{uext}(m_b, \ldots, m_{n-1})(\alpha) = \frac{y_H}{\alpha^{D_H-(n-b)}}$ The verifier computes for itself $y_c = \mathsf{uext}(m'_0, \ldots, m'_{b-a-1})(\alpha)$. By inspection, if $m' = m[a:b]$, then $\mathsf{uext}(m_0, \ldots, m_{n-1}) = \mathsf{uext}(m_0, \ldots m_{a-1}) + X^a \mathsf{uext}(m'_0, \ldots, m'_{b-a-1}) + X^b \mathsf{uext}(m_b, \ldots, m_{n-1})$, so completeness holds.

Lastly, assuming the polynomial commitment scheme used is binding, the additional soundness error incurred by this protocol can be bounded by the Schwartz-Zippel lemma. The last check involves checking that the opening of com (which is a degree-n polynomial) and the computed value y are equal. The Schwartz-Zippel lemma cannot be directly applied to the equality $y_m = \frac{y_H \cdot \alpha^b}{x^{D_H-(n-b)}} + y_c \cdot \alpha^a + \frac{y_L}{x^{D_L-a}}$ because there are terms in the denominator. However, this equality holds (except for probability $1/|\mathbb{F}|$ when $x = 0$) if and only if the same equation multiplied by $x^{\max(D_L-a, D_H-(n-b))}$ holds, which is an equality of polynomials. The degrees of relevant polynomials satisfy $\max(D_L - a, D_H - (n-b)) \leq n$, $deg(f) < n$. The degree of $\mathsf{uext}(m_0, \ldots, m_{a-1}) \cdot X^{D_L-a}, \mathsf{uext}(m_b, \ldots, m_{n-1}) \cdot X^{D_H-(n-b)}$ is at most $2n$, and the degree of X^a, X^b is at most n. When these polynomials are not the same in the case of a dishonest prover, the probability that two polynomials of degree at most $3n$ agree at a random evaluation point occurs with probability at most $\frac{3n}{|\mathbb{F}|}$, and the additional soundness error of this protocol is at most $\frac{3n}{|\mathbb{F}|}$.

Next, we discuss the security of our multilinear polynomial-based redactable signatures. We define PaddedMLE to refer to the construction where blocks of L field elements in the message m are padded with zeros so they fit into blocks whose lengths are powers of 2.

Theorem J.3. *We claim that the prover and verifier algorithms in Figure 6 form a succinct interactive argument of knowledge for the instance-witness relation* $\mathcal{R}^{n,L}_{\mathsf{mlrpcs}}$.

Again, for purposes of security analysis, we interpret this as an interactive protocol where the hashing steps are replaced with commitments sent from the signer to verifier, and a random public challenge from the verifier to the prover. We also assume that the initial polynomial commitment is computed honestly.

The succinctness and public-coin properties follow from the properties of the underlying polynomial commitment scheme and by inspection on the messages sent.

This protocol is complete by Equation 9 and the correctness of the PCS.

Lastly, assuming the polynomial commitment scheme used is binding, the additional soundness error incurred by this protocol is the probability that two distinct multilinear polynomials in $\log n$ variables agree at a random evaluation point, so the soundness error incurred by this protocol for a message of length n is less than or equal to $\frac{\log n}{|\mathbb{F}|}$.

Lastly, we discuss the security of the sumcheck-based redactable signature scheme discussed in Appendix I.2.2.

Theorem J.4. *The sumcheck-based redactable signature scheme in Appendix I.2.2 is a succinct interactive argument of knowledge for the instance witness relation* $\mathcal{R}^n_{\mathsf{mlrsc}}$.

As mentioned in Appendix I.2.2, with $m, m', v(X), f(X), g(X)$ as defined in that section, $f(X)v(X) - g(X)$ is zero everywhere on the Boolean hypercube if and only if $m[a:b] = m'$ because of how we define f, v, and g.

Next, using the results of Section 3.2 in Hyperplonk [20], applying the Zerocheck PIOP to $f(X)v(X) - g(X)$ is a sound and complete interactive protocol for this relation. The polynomial $f(X)v(X) - g(X)$ is log n-variate, with degree at most 2 in each variable.

Since this is just an application of the zero-check PIOP, by the results of Section 3.2 of Hyperplonk, this is a complete and knowledge-sound protocol. The prover requires $O(n)$ $\mathbb{F}$-operations and the verifier requires $O(\log n)$ $\mathbb{F}$-operations.