Eva: Efficient Privacy-Preserving Proof of Authenticity for Lossily Encoded Videos

Zhang, Chengru ; Yang, Xiao; Oswald, David; Ryan, Mark; Jovanovic, Philipp

Citation for published version (Harvard):
Zhang, C, Yang, X, Oswald, D, Ryan, M & Jovanovic, P 2025, Eva: Efficient Privacy-Preserving Proof of Authenticity for Lossily Encoded Videos. in *46th IEEE Symposium on Security and Privacy.* IEEE Symposium on Security and Privacy, IEEE, 46th IEEE Symposium on Security and Privacy, San Francisco, California, United States, 12/05/25.

Eva: Efficient Privacy-Preserving Proof of Authenticity for Lossily Encoded Videos

Chengru Zhang*, Xiao Yang†✉, David Oswald†, Mark Ryan† and Philipp Jovanovic‡
*University of Hong Kong †University of Birmingham ‡University College London

Abstract—**With the increasing usage of fake videos in misinformation campaigns, proving the provenance of an edited video becomes critical, in particular, without revealing the original footage. We formalize the notion and security model of proofs of video authenticity and give the first cryptographic video authentication protocol Eva, which supports lossy codecs and arbitrary edits and is proven secure under well-established cryptographic assumptions. Compared to previous cryptographic methods for image authentication, Eva is not only capable of handling significantly larger amounts of data originating from the complex lossy video encoding but also achieves linear prover time, constant RAM usage, and constant proof size with respect to video size. These improvements have optimal theoretic complexity and are enabled by our two new theoretical advancements of integrating lookup arguments with folding-based incrementally verifiable computation (IVC) and compressing IVC proof efficiently, which may be of independent interest. For our implementation of Eva, we then integrate them with the Nova folding scheme, which we call Loua. As for concrete performance, we additionally utilize various optimizations such as tailored circuit design and GPU acceleration to make Eva highly practical: for a 2-minute HD (1280×720) video encoded in H.264 at 30 frames per second, Eva generates a 448 B proof in about 2.4 hours on consumer-grade hardware at 2.6 µs per pixel, surpassing state-of-the-art cryptographic image authentication schemes by more than an order of magnitude in terms of prover time and proof size.**

1. Introduction

Disinformation campaigns frequently target visual multimedia content, like images and videos, due to their popularity and ease of distribution on social media platforms [1], [2]. This trend has been exacerbated recently by the rapid evolution of (generative) AI tools [3], [4], [5] that enable the manipulation, generation, and dissemination of (fake) multimedia content with a few clicks, presenting a significant challenge to content moderation and fact-checking systems.

To combat maliciously generated multimedia content, the two primary defenses include 1) the *detection* of fake content, by humans [6], [7] or automated methods [8], [9], and 2) the *authentication* of genuine content in which a *prover* tries to convince a *verifier* of the content's provenance by providing some authentication information [10], [11], [12]. Among authentication-based approaches, the Coalition for Content Provenance and Authenticity (C2PA) standard [10] is an industry-wide effort to authenticate multimedia content based on digital signatures.

The issue with these existing approaches is that they either lack flexibility or raise security and privacy concerns. In practice, raw multimedia content often needs to be edited and encoded before publishing, but authentication-based methods [11], [12], [13] typically allow only a limited set of predefined transformations. While C2PA permits arbitrary edits, it requires trusted editing software to sign the transformations, introducing trust assumptions that are difficult to meet, as an attacker may be able to extract the signing key from the software and generate legitimate signatures for any edits. In addition, C2PA's metadata may expose sensitive data that is not intended for disclosure, such as the thumbnail of the original footage. Furthermore, many methods based on detection [6], [7], [8], [9], [14] and authentication [11], [12], [13] are prone to false positives or false negatives with a non-negligible probability, an issue which may be even worse in the presence of active attackers who can bypass these mechanisms by exploiting their vulnerabilities [15], [16]. Cryptographic solutions have been proposed for *image* authentication [17], [18], [19], [20], [21], [22] tackling some of these challenges. Still, the problem of *video* authentication remains largely unaddressed, as it is a significantly harder task mainly due to the following two challenges:

- First, while lossless image encoding is common in practice, video encoding is usually lossy. Consequently, a video prover has to support lossy encoding that involves significant complexities. Otherwise, the verifier cannot recover the edited video that exactly matches the prover's claim because of the information loss caused by encoding. In contrast, for lossless images, the prover can simply prove the honesty of editing without considering encoding, as the edited images can be reconstructed accurately by the verifier.
- Second, videos extend images by adding a time dimension, increasing data sizes significantly. However, authenticating large amounts of (edited) data, which is usually achieved through zkSNARKs, imposes heavy computational and memory costs on the prover. For instance, it takes more than 1 day for the state-of-the-art to prove 1800 HD lossless images (equivalent to a 1-minute HD video at 30 FPS), let alone lossy formats.

In this work, we introduce Eva, the first cryptographic protocol for authentication of lossy-encoded videos. The core protocol works as follows: 1) After recording a video

✉ Corresponding author. Email: x.yang.10@bham.ac.uk

footage $\boldsymbol{V}$, the recorder signs its hash $\mathsf{H}(\boldsymbol{V})$ and produces signature σ. 2) After the prover edits $\boldsymbol{V}$ and encodes the edited video $\boldsymbol{V}'$ to obtain ζ, a proof π is generated, showing that σ is a valid signature on $\mathsf{H}(\boldsymbol{V})$ and that $\boldsymbol{V}$ is honestly transformed into ζ. 3) When the verifier receives ζ and π, it can verify the authenticity of ζ even without access to $\boldsymbol{V}$.

To address the first challenge, a naive solution is to prove that the edited video $\boldsymbol{V}'$ and the video $\widetilde{\boldsymbol{V}}$ reconstructed from the encoded bitstream ζ are "similar", but it is difficult to define a metric to quantify such similarity without introducing false positives or false negatives. Proving the honesty of video encoding is thus inevitable to achieve negligible error rates. However, expressing the highly complex encoding process in zkSNARK circuits is intricate and costly. Our key insight is that, even though the video is encoded in a lossy format, certain intermediate data remains identical between encoding and decoding. Such data can be accurately recovered by the verifier when decoding, and thus the prover no longer needs to prove its validity. In fact, by feeding such data as public inputs during proof verification, we can skip proving the most complicated parts of the encoding process (e.g., inter-frame prediction and entropy coding) and instead focus on manageable components, thereby significantly reducing the difficulty of circuit construction.

To address the second challenge, we exploit the highly repetitive structure of videos and video processing algorithms: they are usually based on *macroblocks*, *i.e.*, small and fixed-size blocks of pixels that can be processed independently. This allows us to leverage *Incrementally Verifiable Computation* (IVC) [23] constructed from *folding schemes* [24], [25], [26], [27], [28] to reduce the circuit size and memory requirements. In each IVC step, the prover only needs to 1) generate an *incoming proof* of the honest editing and encoding for a few macroblocks, and then 2) accumulate it into the *running proof*. Finally, we also leverage *lookup arguments* [29], [30], [31], [32] to avoid expensive bit operations in the arithmetic circuits for video processing.

1.1. Contribution

Below we summarize the contributions of our work.

Eva**: Efficient Video Authentication.** As the core contributions of our work, we formalize the notion of *proofs of video authenticity* along with its security model, and we propose a secure and efficient construction, Eva. Eva is not only the first cryptographic protocol for videos, but also the first cryptographic scheme that takes lossy codecs into account.

Due to the complexity of video codecs and the large size of videos, it is unrealistic to naively prove video authenticity in-circuit. Eva makes it not only feasible but also efficient, yielding linear prover time and constant prover RAM and proof size in video size, and it is concretely performant even on consumer-grade hardware.

- We avoid proving complex steps in the encoding process by leveraging shared data between encoders and decoders, hence simplifying our circuits for encoding.
- Taking advantage of the repetitive structure, videos are proven block-wise with manageable costs per step using IVC. This makes Eva capable of handling arbitrarily large video files with a constant memory footprint.
- To further reduce circuit size, we use both general techniques, e.g., lookup arguments and non-deterministic advice, and tailored approaches, such as efficient handling of branches-based on dynamic conditions.

In addition, Eva naturally supports lossless encoding as well, if we skip the encoding circuits and only prove editing.

Moreover, Eva is proven secure in our model under well-established cryptographic assumptions, providing soundness against attackers and zero-knowledge of the original video except with negligible probability. By incorporating Eva into the C2PA standard, we can not only improve the security of C2PA by eliminating the trust assumptions on editing software but also provide better privacy by hiding the original footage from the verifier.

Paradigms for improving folding-based IVC. As theoretical contributions that might be of independent interest, we present two general paradigms to enhance folding-based IVC. The first paradigm integrates LogUp [33], an efficient lookup argument [29], with arbitrary folding-based IVC. The second paradigm constructs a *decider* that compresses the proof of folding-based IVC into a constant-size and zero-knowledge one via *commit-and-prove SNARKs* (CP-SNARKs) [34], without introducing vector-to-polynomial conversions or evaluation arguments as in [24], [35].

By applying our paradigms to a popular folding scheme Nova [24], we introduce a variant Loua, which offers efficiency improvements over both the original Nova and its design in the `sonobe` folding library [35]. With Loua at the core of Eva, we achieve a significant reduction in the number of constraints for video encoding and editing by substituting expensive bitwise operations with cheap lookups.

Implementation and evaluation. We implement Eva and Loua upon the `sonobe` library, with optimizations such as GPU computing. We show compatibility with H.264 [36] and employ a circuit-friendly hash function Griffin [37] and Schnorr signature [38], but our modularized design allows one to choose different schemes and support other lossy, or lossless formats (by omitting the encoding circuit) for videos or images and to achieve full compliance with C2PA.

Even with larger data size and an additional encoding process, Eva yields $> 10\times$ improvements in prover time and proof size over state-of-the-art constructions for images. For a 2-minute HD H.264 video at 30 frames per second, Eva generates a proof in ~ 2.4 hours on a consumer-grade desktop. During IVC proof generation, the RAM usage is kept at a constant ~ 10 GB, and IVC proof compression requires $50 \sim 60$ GB of RAM. The final proofs have a constant size of 448 bytes and can be verified by resource-constrained devices like mobile phones or blockchain validators.

1.2. Related Work

By regarding videos as a generalization of images, we summarize the comparison of Eva to cryptographic protocols for image authentication in Table 1. For details of performance evaluation, we refer the reader to Section 6.

TABLE 1: Comparison between Eva and cryptographic protocols for image authentication.[a]

	Format	Compression	Editing Operations	Prover time	Prover RAM	Proof size	Max dimensions[b]
PhotoProof [17]	Image	Lossless	Arbitrary	$O(P^3 \log P)$ ($<$ 18676 μs/px)	$O(P)$	$O(1)$ (2.67 KB)	128 × 128 $< \infty$
VIR [18]	Image	Lossless	Masking	$O(P \log P)$ ($\sim$ 16 μs/px)	$O(P)$	$O(1)$ (223 B)	3840 × 2160 $< \infty$
ZK-IMG [19]	Image	Lossless	Arbitrary	$O(P \log P)$ ($>$ 355 μs/px	$O(P)$	$O(\log P)$ ($\sim$10 KB)	1280 × 720 $< \infty$
VIMz [20]	Image	Lossless	Arbitrary	$O(P)$ ($\sim$ 167 μs/px)	$O(N)$	$O(\log^2 N)$ ($\sim$10 KB)	3840 × 2160 $< \infty$
VerITAS [21]	Image	Lossless	Arbitrary	$O(P \log P)$ ($\sim$ 95 μs/px)	$O(P)$	$O(\log^2 P)$ ($\sim$100 KB)	6632 × 4976 $< \infty$
TilesProof [22]	Image	Lossless	Arbitrary	$O(P \log(P/T))$ ($\sim$ 62 μs/px)	$O(P/T)$	$O(T)$ ($\sim$10 KB)	6000 × 4000 $< \infty$
Eva (this work)	Video	Lossy (H.264) Lossless	Arbitrary	$O(P)$ ($\sim$ 2.6 μs/px)	$O(1)$	$O(1)$ (448 B)	1280 × 720 × 3600 ∞

[a] Asymptotic complexity is measured w.r.t. the number of pixels $P = MNL$ and the number of tiles T (required by [22]), where M is the height, N is the width, and L is the time. Inside the parentheses is the concrete performance evaluated on our machine when possible. If the source code is unavailable, we quote the original authors' results and indicate the estimated results on our machine using $>$ and $<$.

[b] Each cell displays the empirical (upper value) and theoretical (lower value) maximum dimensions. ∞ refers to unlimited dimensions, while $< \infty$ means that unlimited dimensions are unsupported (due to bounded RAM).

PhotoProof [17] is a pioneering work in this direction that uses *Proof-Carrying Data* [39] to offer authenticity of edited images. Due to the high computational cost of prover, it only supports tiny images. In [18], Ko et al. propose VIR, which utilizes CP-SNARKs [34] to generate constant-size proofs of redaction on images (masking secret parts with black tiles). VIR significantly reduces the prover time while supporting much larger images. Built upon halo2 [40], a more efficient proof system, ZK-IMG [19] also has faster prover than PhotoProof while maintaining support for arbitrary editing operations. More recent schemes include VIMz [20], VerITAS [21], and TilesProof [22]. We note that these works share several common ideas with us, e.g., VIMz also employs folding schemes to reduce prover RAM costs, and VerITAS as well utilizes lookup arguments to improve prover time. Our work integrates these general techniques with video processing algorithms, and also introduces customized IVC, tailored circuit design, and dedicated optimizations to maximize the efficiency of Eva. Consequently, compared to all other protocols, Eva achieves optimal time and space complexity and delivers the best concrete prover and verifier performance. Further, all previous works are limited to lossless images, while our work supports lossy video codecs, tackling a much greater challenge.

1.3. Overview of Eva

We provide an overview of Eva with a motivating example and explain a video's life cycle in our scheme. Consider an on-site whistleblower Alice who wants to record and publish a video to expose illegal activities. Before publishing the video, Alice wishes to blurs her face for privacy concerns. At the same time, she also aims to convince the viewers that blurring is the only edit that she has made, and that this operation is done on an authentic footage.

Eva can prove the video's provenance while preserving Alice's privacy. This is done by performing the steps below. An illustrated version can be found in Figure 1.

Setup. At the beginning of Eva, device manufacturers generate keys for a signature scheme and embed them into the hardware root-of-trust of recording devices. Here, we assume that the manufacturers are able to obtain certificates (e.g., from C2PA) for their keys. They also produce parameters for proving and verifying the authenticity of videos.

Record and sign. With such a recording device, Alice now captures a video (raw footage), after which the device creates a signature on the footage (along with metadata such as timestamp, location, etc.) under the embedded signing key. The signature ensures that the footage is authentic, or more specifically, is recorded by a C2PA-certified device.

Edit and prove. Alice then edits the footage and blurs her face. The video processing software usually encodes the edited video and outputs a compressed video stream with much smaller size, which is thus more suitable for publishing. After that, Alice generates a succinct zero-knowledge proof through Eva, showing that the final video stream is a blurred-then-encoded version of an authentic video signed by a certified recording device. She can now upload the video stream as well as the proof to her website, together with a claim that she only performed the blurring operation.

Verify. Using the verification algorithm of Eva, a visitor of Alice's website can examine the video stream's provenance by checking the proof against the video and the claimed edits. If the proof holds, then the visitor is convinced that the video is a blurred version of an authentic footage, which is taken by a certified device at a specific time and location.

2. Preliminaries

Notations. In this paper, $y := F(x)$ denotes the output of a deterministic algorithm F on input x. For a randomized algorithm F, we write $y \leftarrow F(x)$, or $y := F(x; r)$ when it is supplied with external randomness r. With security parameter λ, a negligible function in λ is denoted by $\varepsilon(\lambda)$.

Vectors and matrices are denoted by boldface italic lowercase and uppercase letters, respectively. $\boldsymbol{x}[i, j]$ is the

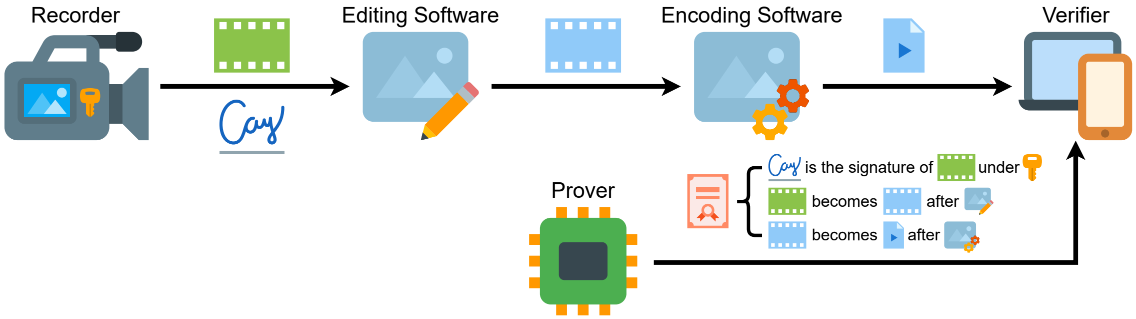


Figure 1: Overview of the Eva protocol. The recorder produces a raw footage and signs it with its embedded signing key. The footage is then edited and encoded into a video stream. The prover generates a proof attesting to the authenticity of the video. The verifier can check the proof of authenticity against the video stream.

subvector of $\boldsymbol{x}$ from index i to j, and $\boldsymbol{X}[i,j;k,l]$ is the submatrix of $\boldsymbol{X}$ from row i to j and column k to l, inclusive.

We consider a *half-pairing cycle* of elliptic curve groups $(\mathbb{G},\widehat{\mathbb{G}},\mathbb{G}_T),\mathbb{H}$, where $\mathbb{G}$ and $\mathbb{H}$ form a 2-cycle, and $(\mathbb{G},\widehat{\mathbb{G}},\mathbb{G}_T)$ is a pairing-friendly group. In this cycle, $\mathbb{F}_q$, the base field of $\mathbb{G}$, is also the scalar field of $\mathbb{H}$; and $\mathbb{F}_p$, the scalar field of $\mathbb{G}$, is also the base field of $\mathbb{H}$.

Algorithms are written in pseudocode. “**assert** *cond*” denotes the operation that returns 0 when the condition *cond* is not satisfied and does nothing otherwise. Its in-circuit counterpart, “**enforce** $x = y$”, adds a constraint for enforcing equality between x and y to the constraint system.

Cryptographic Primitives. We rely on collision-resistant hash functions H, ρ, and ϱ, an EUF-CMA secure signature scheme $\mathsf{Sig} = (\mathsf{Sig}.\mathcal{K},\mathsf{Sig}.\mathcal{S},\mathsf{Sig}.\mathcal{V})$, and a binding and hiding commitment scheme $\mathsf{CM} = (\mathsf{CM}.\mathcal{K},\mathsf{CM}.\mathcal{C},\mathsf{CM}.\mathcal{V})$.

Here, we consider ρ as a random oracle in the random oracle model. Further, we define $\varrho(x;r) = \rho(x \mid\mid r)$ as a hiding hash function. The signing key and verification key in Sig are denoted by sk and vk, respectively. The commitment key in CM is denoted by ck. For simplicity, we omit the randomness in ϱ and CM from the notation.

We assume familiarity with the notion of zk-SNARKs. A zero-knowledge *commit-and-prove SNARK* (CP-SNARK) [34], [41] $\mathsf{ZKCP} = (\mathsf{ZKCP}.\mathcal{K},\mathsf{ZKCP}.\mathcal{P},\mathsf{ZKCP}.\mathcal{V})$ for relation R is a zkSNARK for the commit-and-prove relation $R^{\mathsf{cp}}\left((\boldsymbol{x},\boldsymbol{c}),((\boldsymbol{v}_i)_{i=0}^{\ell-1},\boldsymbol{\omega})\right)$, which holds iff. $R\left(\boldsymbol{x},((\boldsymbol{v}_i)_{i=0}^{\ell-1},\boldsymbol{\omega})\right) = 1 \wedge \bigwedge_{i\in[0,\ell-1]} \mathsf{CM}.\mathcal{V}(\mathsf{ck},c_i,\boldsymbol{v}_i) = 1$, *i.e.*, the statements $\boldsymbol{x}$ and the witnesses $((\boldsymbol{v}_i)_{i=0}^{\ell-1},\boldsymbol{\omega})$ satisfy R, and each c_i is a commitment to a part of witnesses $\boldsymbol{v}_i$.

For a vector of *queries* $\boldsymbol{\alpha} = (\alpha_i)_{i=0}^{\mu-1}$ and a *lookup table* $\boldsymbol{\tau} = (\tau_j)_{j=0}^{\nu-1}$, a lookup argument [29], [30], [31], [32] is a zkSNARK for the lookup relation $R^{\mathsf{lookup}}(\boldsymbol{\tau},\boldsymbol{\alpha})$, which holds iff. $\{\alpha_i\}_{i=0}^{\mu-1} \subseteq \{\tau_j\}_{j=0}^{\nu-1}$.

A non-interactive folding scheme [24] $\mathsf{NIFS} = (\mathsf{NIFS}.\mathcal{G},\mathsf{NIFS}.\mathcal{K},\mathsf{NIFS}.\mathcal{P},\mathsf{NIFS}.\mathcal{V})$ folds two instances for relation R into a single instance such that the correctness of the folded instance implies that of the original ones.

Incrementally verifiable computation [23] $\mathsf{IVC} = (\mathsf{IVC}.\mathcal{G},\mathsf{IVC}.\mathcal{K},\mathsf{IVC}.\mathcal{P},\mathsf{IVC}.\mathcal{V})$ allows one to verify the repeated execution of a function $\mathcal{F}$, dubbed *step function*. Specifically, $\mathsf{IVC}.\mathcal{P}$ can generate a proof π_i that the current state $\boldsymbol{z}_i$ is the result of i invocations of $\mathcal{F}$ starting from an initial state $\boldsymbol{z}_0$, given the proof π_{i-1} attesting to $\boldsymbol{z}_{i-1}$.

3. Proofs of Video Authenticity

We formalize *proofs of video authenticity*, a category of video authentication protocols that are provably secure. We begin by describing the data types and operations involved, followed by the algorithm definition and security properties.

3.1. Data Types and Operations

Video. There are two forms of *video* data: the raw video $\boldsymbol{V}$ and the video stream ζ. A *raw video* is for being displayed on a screen or edited by video processing software. It is composed of a series of *frames* ordered by time. Each frame is a still image described as a 2D matrix of *pixels*. It is common to use the YCbCr color space in video processing, where a pixel is represented by a luma component Y and two chroma components Cb, Cr, each with 8 bits.

Moreover, in video processing, a frame is usually partitioned into *macroblocks* of size 16×16, which contains 16×16 bytes for Y and 8×8 bytes for both Cb and Cr, due to *subsampling*. Formally, we define a macroblock as $\boldsymbol{X} := (\boldsymbol{X}^{\mathsf{Y}},\boldsymbol{X}^{\mathsf{Cb}},\boldsymbol{X}^{\mathsf{Cr}}) \in \mathcal{B}$, where $\mathcal{B}$ is the set of all possible macroblocks, *i.e.*, $\mathcal{B} := \mathbb{Z}_{2^8}^{16\times16} \times \mathbb{Z}_{2^8}^{8\times8} \times \mathbb{Z}_{2^8}^{8\times8}$. In consequence, for a video with L frames, each of which has M rows and N columns, we write $\boldsymbol{V} := (\boldsymbol{X}_i)_{i=0}^{M/16\times N/16\times L-1} \in \mathcal{B}^{M/16\times N/16\times L}$.

Due to the large size, a raw video is compressed into a sequence of bits ζ, or formally, a *video stream*, when being transmitted over the network or stored in a file to reduce communication and storage costs.

Encoding and Decoding. In video codecs, a raw video $\boldsymbol{V}$ is converted to a video stream ζ by the *encoder*, whereas the *decoder* reconstructs a video $\widetilde{\boldsymbol{V}}$ from a video stream ζ. The codec is *lossless* if the decoder can reconstruct the original video exactly, i.e., $\boldsymbol{V} = \widetilde{\boldsymbol{V}}$, and *lossy* if some information is discarded, exchanging quality for a smaller video stream.

Figure 2 gives the general workflow of macroblock-based video codecs [36], [42], [43]. We can observe that, on input an *original macroblock* $\boldsymbol{X}$, the block-wise encoder $\mathcal{E}$ generates a *prediction macroblock* $\boldsymbol{P}$ with some reference information ref and computes the residual macroblock $\boldsymbol{R}$ by subtracting $\boldsymbol{P}$ from $\boldsymbol{X}$. Then $\boldsymbol{R}$ is converted to *transformed coefficients* $\boldsymbol{Y}$ and then *quantized coefficients* $\boldsymbol{Z}$ via transform and quantization. Finally, $\boldsymbol{Z}$ is compressed into ζ by entropy encoding. The decoder $\mathcal{D}$ essentially “inverts” the encoder’s process. By applying $\mathcal{E}$ to all macroblocks of a video $\boldsymbol{V}$, we can obtain the encoded video stream $\zeta := \mathcal{E}(\boldsymbol{V}, \mathsf{param}^{\mathcal{E}})$. The decoded video can be computed as $\boldsymbol{V}' := \mathcal{D}(\zeta, \mathsf{param}^{\mathcal{E}})$. Further, we assume that one can extract intermediate data from $\mathcal{E}$ and $\mathcal{D}$, such as the prediction macroblock $\boldsymbol{P}$ and quantized coefficients $\boldsymbol{Z}$.

Metadata. Metadata meta is a set of information associated with the video, such as the recording device ID, the location and time of recording. We assume that meta is immutable.

Editing. A block-wise editing operation Δ is defined as $\Delta : \mathcal{B} \times \{0,1\}^* \to \mathcal{B}$, which takes a macroblock $\boldsymbol{X}$ and some editing parameters $\mathsf{param}^{\Delta} \in \{0,1\}^*$ as input, edits $\boldsymbol{X}$, and outputs the edited macroblock $\boldsymbol{X}'$.

3.2. Algorithm and Security Definitions

A proof of video authenticity involves four parties: the *trusted party*, the *recorder*, the *prover*, and the *verifier*.

The trusted party (e.g., manufacturer) runs the key generation algorithms $\mathcal{K}_\Sigma$ and $\mathcal{K}_\Pi$, where $\mathcal{K}_\Sigma$ generates signing keys for the recorders, and $\mathcal{K}_\Pi$ produces necessary parameters for proof generation and verification. The signing keys are then securely provisioned to the recorders and are safely protected using mechanisms such as secure enclaves.

The recorder (e.g., camcorder) records a video, generates the metadata, and runs the recording algorithm $\mathcal{R}$, which signs the video and the metadata under the signing key.

The prover (e.g., content creator) edits and encodes the original video, publishes the processed video, and runs the proof generation algorithm $\mathcal{P}$ to get a proof of authenticity.

The verifier (e.g., website visitor) checks if the proof is valid w.r.t. the video using the verification algorithm $\mathcal{V}$.

Now we formally define the algorithms discussed above in a proof of video authenticity.

Definition 1 (Proof of Video Authenticity). *A proof of video authenticity is defined as* $\mathsf{VA} = (\mathcal{K}_\Sigma, \mathcal{K}_\Pi, \mathcal{R}, \mathcal{P}, \mathcal{V})$*:*

- $\mathcal{K}_\Sigma(1^\lambda) \to (\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma)$
  $\mathcal{K}_\Pi(1^\lambda) \to (\mathsf{pk}_\Pi, \mathsf{vk}_\Pi)$
  Both key generation algorithms $\mathcal{K}_\Sigma$ and $\mathcal{K}_\Pi$ take as input security parameter 1^λ. $\mathcal{K}_\Sigma$ outputs a pair of secret signing key sk_Σ *and public signature verification key* vk_Σ*, and $\mathcal{K}_\Pi$ outputs a pair of public proving key* pk_Π *and public proof verification key* vk_Π*. $\mathcal{K}_\Pi$ also returns a secret trapdoor* td *that is omitted from the notation for simplicity but is used in security definitions.*
- $\mathcal{R}(\mathsf{sk}_\Sigma, \boldsymbol{V}, \mathsf{meta}) \to \sigma$
  The recording algorithm $\mathcal{R}$ takes as input signing key sk_Σ*, video $\boldsymbol{V}$ and its metadata* meta*, and outputs a signature σ on $\boldsymbol{V}$ and* meta.
- $\mathcal{P}(\mathsf{pk}_\Pi, \mathsf{vk}_\Sigma, \boldsymbol{V}, \mathsf{meta}, \mathsf{param}, \sigma) \to (\zeta, \pi)$
  The proof generation algorithm $\mathcal{P}$ takes as input proving key pk_Π*, signature verification key* vk_Σ*, video $\boldsymbol{V}$, metadata* meta*, editing and encoding parameters* $\mathsf{param} = (\mathsf{param}^{\Delta}, \mathsf{param}^{\mathcal{E}})$*, and signature σ. It outputs a video stream ζ and a proof π that attests to 1) the honesty of the editing and encoding process from $\boldsymbol{V}$ to ζ under* param*, and 2) the validity of σ on $(\boldsymbol{V}$,* $\mathsf{meta})$ *under* vk_Σ.
- $\mathcal{V}(\mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \zeta, \mathsf{meta}, \mathsf{param}, \pi) =: b$
  The verification algorithm $\mathcal{V}$ takes as input proof verification key vk_Π*, signature verification key* vk_Σ*, processed video stream ζ and its metadata* meta'*, editing and encoding parameters* param*, and proof π, and outputs a bit b indicating if the proof is valid for ζ,* meta *and* vk_Σ.

Now we formalize the security of VA. Consider the relation $R^{\mathsf{VA}}(\boldsymbol{x}, \boldsymbol{w})$ for the authenticity of a video, where $\boldsymbol{x} = (\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), \boldsymbol{w} = (\sigma, \boldsymbol{V})$. For a signature scheme Sig, an editing operation Δ, and an encoder $\mathcal{E}$, $R^{\mathsf{VA}}(\boldsymbol{x}, \boldsymbol{w}) = 1$ iff.

$$\mathsf{Sig}.\mathcal{V}(\mathsf{vk}_\Sigma, \sigma, (\boldsymbol{V}, \mathsf{meta})) = 1 \land \zeta = \mathcal{E}(\Delta(\boldsymbol{V}, \mathsf{param}^{\Delta}), \mathsf{param}^{\mathcal{E}})$$

The security of VA is defined below, which can be regarded as the security of zkSNARKs for R^{VA}.

COMPLETENESS. Completeness holds if for every video $\boldsymbol{V}$, metadata meta, and editing and encoding parameters param,

$$\Pr\left[\begin{array}{l} (\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma) \leftarrow \mathcal{K}_\Sigma(1^\lambda) \\ (\mathsf{pk}_\Pi, \mathsf{vk}_\Pi) \leftarrow \mathcal{K}_\Pi(1^\lambda) \\ \sigma \leftarrow \mathcal{R}(\mathsf{sk}_\Sigma, \boldsymbol{V}, \mathsf{meta}) \\ (\zeta, \pi) \leftarrow \mathcal{P}(\mathsf{pk}_\Pi, \mathsf{vk}_\Sigma, \boldsymbol{V}, \mathsf{meta}, \mathsf{param}, \sigma) \\ R^{\mathsf{VA}}((\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), (\sigma, \boldsymbol{V})) = 1 : \\ \qquad \mathcal{V}(\mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \zeta, \mathsf{meta}, \mathsf{param}, \pi) = 1 \end{array}\right] = 1$$

KNOWLEDGE SOUNDNESS. Knowledge soundness holds if for every p.p.t. adversary $\mathcal{A}$, there is a p.p.t. extractor Ext such that for all input randomness r,

$$\Pr\left[\begin{array}{l} (\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma) \leftarrow \mathcal{K}_\Sigma(1^\lambda) \\ (\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{td}) \leftarrow \mathcal{K}_\Pi(1^\lambda) \\ (\zeta, \mathsf{meta}, \mathsf{param}, \pi) := \mathcal{A}(\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{vk}_\Sigma; r) \\ (\sigma, \boldsymbol{V}) := \mathsf{Ext}(\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \mathsf{td}; r) \\ \mathcal{V}(\mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \zeta, \mathsf{meta}, \mathsf{param}, \pi) = 1 : \\ \qquad R^{\mathsf{VA}}((\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), (\sigma, \boldsymbol{V})) = 0 \end{array}\right] \le \varepsilon(\lambda)$$

ZERO-KNOWLEDGE. Optionally, VA may satisfy the zero-knowledge property, which holds if there exists a simulator Sim such that for every p.p.t. distinguisher $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} (\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma) \leftarrow \mathcal{K}_\Sigma(1^\lambda) \\ (\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{td}) \leftarrow \mathcal{K}_\Pi(1^\lambda) \\ ((\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), (\sigma, \boldsymbol{V})) \leftarrow \mathcal{A}(\mathsf{vk}_\Sigma, \mathsf{pk}_\Pi, \mathsf{vk}_\Pi) \\ (\cdot, \pi) \leftarrow \mathcal{P}(\mathsf{pk}_\Pi, \mathsf{vk}_\Sigma, \boldsymbol{V}, \mathsf{meta}, \mathsf{param}, \sigma) : \\ \hfill \mathcal{A}(\pi) = 1 \end{array}\right]$$

$$\approx \Pr\left[\begin{array}{l} (\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma) \leftarrow \mathcal{K}_\Sigma(1^\lambda) \\ (\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{td}) \leftarrow \mathcal{K}_\Pi(1^\lambda) \\ ((\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), (\sigma, \boldsymbol{V})) \leftarrow \mathcal{A}(\mathsf{vk}_\Sigma, \mathsf{pk}_\Pi, \mathsf{vk}_\Pi) \\ \pi \leftarrow \mathsf{Sim}(\mathsf{td}, \mathsf{pk}_\Pi, \mathsf{vk}_\Sigma, \mathsf{meta}, \mathsf{param}, \zeta) : \\ \hfill \mathcal{A}(\pi) = 1 \end{array}\right]$$

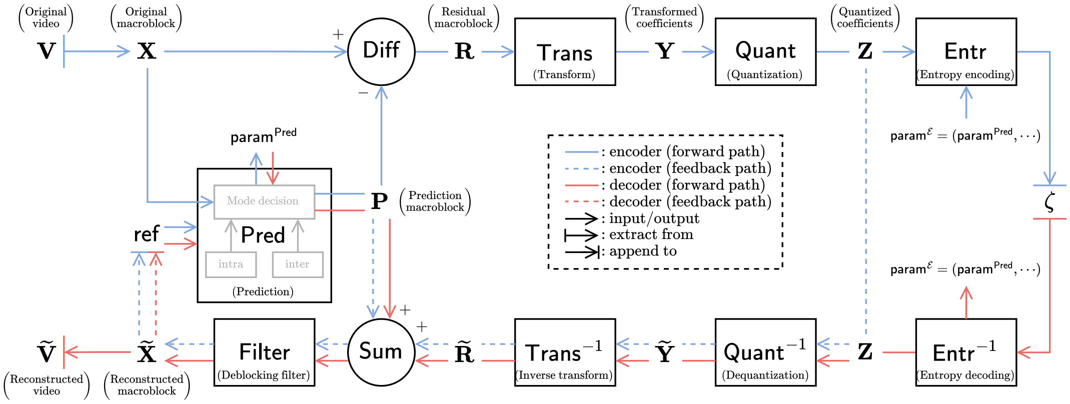


Figure 2: Block diagram of a macroblock-based video codec.

SUCCINCTNESS. Optionally, VA may produce succinct proofs, if for every video $\boldsymbol{V}$ of dimension $M \times N \times L$, the proof π for $\boldsymbol{V}$ is of size $|\pi| = \mathrm{poly}(\lambda)\,\mathrm{polylog}(MNL)$.

4. Improving Folding-Based IVC

What makes Eva an efficient and succinct zero-knowledge proof of video authenticity is a folding-based IVC scheme that supports efficient lookups and proof compression. These capabilities are enabled by our general, scheme-agnostic paradigms for folding-based IVC. The first paradigm incorporates LogUp [33], an efficient lookup argument, into any folding-based IVC scheme. The second paradigm constructs a decider based on CP-SNARKs that compresses IVC proofs into a fully succinct and zero-knowledge proof of constant size. A concrete instantiation Loua is obtained by applying them to Nova [24]-based IVC.

4.1. Paradigm 1: IVC with Lookup Arguments

We first give the high-level idea behind our paradigm for supporting lookup arguments in folding-based IVC, where we consider an IVC scheme IVC constructed from a folding scheme NIFS. In each step of IVC, we first collect $\boldsymbol{q}$, the queries to the lookup table, from the execution of the step function $\mathcal{F}$. Then, in addition to folding the NIFS instances, we also fold $\overline{Q}$, the commitment to $\boldsymbol{q}$. Finally, we create an *augmented step circuit* $\mathcal{F}^{\mathsf{aug}}$ for $\mathcal{F}$ that additionally verifies the lookup relation for $\boldsymbol{q}$ against the exact $\overline{Q}$, thereby linking the folding scheme with the lookup argument.

Existing folding-based IVC. To elaborate on the intuition above, we quickly review the construction of folding-based IVC. Given a folding (accumulation) scheme NIFS, the compilers proposed in [24], [44] are able to convert it to an IVC scheme IVC. Both folding-to-IVC compilers are conceptually similar: an IVC for step function $\mathcal{F}$ internally utilizes NIFS for an *augmented step circuit* $\mathcal{F}^{\mathsf{aug}}$, which 1) performs the current execution of $\mathcal{F}$, and 2) verifies the previous folding proof. Let $\boldsymbol{w}$ and $\boldsymbol{x}$ be the witness and statement (public I/O) of the circuit $\mathcal{F}^{\mathsf{aug}}$. WLOG, we adopt the notations in [24] and assume $\boldsymbol{w}$ is contained in the folding witness $\mathbb{W}$, and $\boldsymbol{x}$ is in the folding instance $\mathbb{U}$.

Specifically, for a step circuit $\mathcal{F}$, the corresponding augmented step circuit $\mathcal{F}^{\mathsf{aug}}$ is associated with an *incoming* instance-witness pair $(\mathbb{u}, \mathbb{w})$ for NIFS. There is also a *running* instance-witness pair $(\mathbb{U}, \mathbb{W})$, which absorbs the incoming one in each step of recursive computation. For instance, in the i-th step, $(\mathbb{u}_i, \mathbb{w}_i)$ represents the execution of $\mathcal{F}^{\mathsf{aug}}$ in the $(i-1)$-th step, while $(\mathbb{U}_i, \mathbb{W}_i)$ is the accumulation of all previous $(\mathbb{u}_j, \mathbb{w}_j)$ for $j \in [0, i-2]$ and thus represents all previous $i-1$ invocations of $\mathcal{F}^{\mathsf{aug}}$.

Given $(\mathbb{u}_i, \mathbb{w}_i)$ and $(\mathbb{U}_i, \mathbb{W}_i)$, the prover IVC.$\mathcal{P}$ folds both pairs, producing a running instance-witness pair $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1})$ that represents all previous i invocations of $\mathcal{F}^{\mathsf{aug}}$. Next, IVC.$\mathcal{P}$ performs the i-th execution of $\mathcal{F}^{\mathsf{aug}}$, whose corresponding $(\mathbb{u}_{i+1}, \mathbb{w}_{i+1})$ is stored for the $(i+1)$-th step.

Here, the i-th execution of $\mathcal{F}^{\mathsf{aug}}$ checks 1) $\mathcal{F}(\boldsymbol{z}_i; \mathsf{aux}_i) = \boldsymbol{z}_{i+1}$, i.e., $\boldsymbol{z}_i$, the state of $\mathcal{F}$ in step i, is correctly updated, 2) NIFS.$\mathcal{V}(\mathsf{vk}, \mathbb{U}_i, \mathbb{u}_i, \overline{T}) = \mathbb{U}_{i+1}$, i.e., $\mathbb{U}_{i+1}$ is the correct folding of $\mathbb{U}_i$, $\mathbb{u}_i$, and 3) $\mathbb{u}_i.\boldsymbol{x} = \varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i)$, where $\mathbb{u}_i.\boldsymbol{x}$ is the statement of $\mathcal{F}^{\mathsf{aug}}$ in the $(i-1)$-th step. Further, $\mathcal{F}^{\mathsf{aug}}$ outputs $h := \varrho(\mathbb{U}_{i+1}, i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1})$, which will be included in $\mathbb{u}_{i+1}.\boldsymbol{x}$, i.e., the statement of $\mathcal{F}^{\mathsf{aug}}$ in the i-th step.

Support lookup in folding-based IVC. Now we discuss how to equip any folding-based IVC with lookup arguments. In our paradigm, we consider $\boldsymbol{\tau} = (\tau_j)_{j=0}^{\nu-1}$, a read-only lookup table with ν entries. We assume during the execution of $\mathcal{F}$, μ queries $\boldsymbol{\alpha} = (\alpha_i)_{i=0}^{\mu-1}$ are made to $\boldsymbol{\tau}$. For the j-th table entry τ_j, we count o_j, the number of τ_j's occurrences in the query vector $\boldsymbol{\alpha}$. The folding witness $\mathbb{W}$ now contains an extra term $\boldsymbol{q} = ((\alpha_i)_{i=0}^{\mu-1}, (o_j)_{j=0}^{\nu-1})$, and its commitment $\overline{Q} = \mathsf{CM}.\mathcal{C}(\mathsf{ck}, \boldsymbol{q})$ is included in the folding instance $\mathbb{U}$.

Having defined $\mathbb{W}$ and $\mathbb{U}$, we build NIFS$^{\mathsf{lookup}}$, a lookup-friendly version of NIFS, whose prover and verifier additionally computes the random linear combination of $\mathbb{U}_1.\overline{Q}$ and $\mathbb{U}_2.\overline{Q}$. The prover also folds $\mathbb{W}_1.\boldsymbol{q}$ and $\mathbb{W}_2.\boldsymbol{q}$ analogously. The construction of NIFS$^{\mathsf{lookup}}$ is illustrated in Algorithm 1. Note that $\boldsymbol{q}$ does not affect the computation of error terms and proofs in NIFS, because it is just a syntactic sugar that denotes a special type of witness $\boldsymbol{w}$ to the relation R, and can be regarded as a plain witness outside the context of lookup arguments. In fact, if $\mathbb{U}$ already contains the commitment to $\boldsymbol{w}$ (e.g., in [24], [25], [26], [27], [28]), our approach can be viewed as the separation of $\boldsymbol{q}$ from $\boldsymbol{w}$.

Algorithm 1: $\mathsf{NIFS}^{\mathsf{lookup}}$

```
Fn $\mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{G}(1^\lambda)$:
    return $\mathsf{pp} \leftarrow \mathsf{NIFS}.\mathcal{G}(1^\lambda)$
Fn $\mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{K}(\mathsf{pp}, R)$:
    return $(\mathsf{pk}, \mathsf{vk}) := \mathsf{NIFS}.\mathcal{K}(\mathsf{pp}, R)$
Fn $\mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2))$:
    $(\mathbb{U}, \mathbb{W}, \overline{T}) \leftarrow \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2))$
    $r := \rho(\mathsf{tr})$    ▷ tr is the transcript between $\mathcal{P}$ and $\mathcal{V}$
    $\mathbb{U}.\overline{Q} := \mathbb{U}_1.\overline{Q} + r \cdot \mathbb{U}_2.\overline{Q}$
    $\mathbb{W}.\boldsymbol{q} := \mathbb{W}_1.\boldsymbol{q} + r \cdot \mathbb{W}_2.\boldsymbol{q}$
    return $(\mathbb{U}, \mathbb{W}, \overline{T})$
Fn $\mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T})$:
    $\mathbb{U} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T})$
    $r := \rho(\mathsf{tr})$    ▷ tr is the transcript between $\mathcal{P}$ and $\mathcal{V}$
    $\mathbb{U}.\overline{Q} := \mathbb{U}_1.\overline{Q} + r \cdot \mathbb{U}_2.\overline{Q}$
    return $\mathbb{U}$
```

Next, we construct $\mathsf{IVC}^{\mathsf{lookup}}$ from $\mathsf{NIFS}^{\mathsf{lookup}}$. Recall that IVC for $\mathcal{F}$ internally utilizes NIFS for an augmented step circuit $\mathcal{F}^{\mathsf{aug}}$. This is also the case in our construction. Inspired by `gnark` [45], which incorporates LogUp into Groth16 [46] and Plonk [47], our augmented step circuit $\mathcal{F}^{\mathsf{lookup}}$ (Circuit 2) additionally checks the set inclusion identity of LogUp [33, Lemma 5] in-circuit.

Suppose $\mathcal{F}$ makes queries $\boldsymbol{\alpha} = (\alpha_i)_{i=0}^{\mu-1}$ to a lookup table with entries $\boldsymbol{\tau} = (\tau_j)_{j=0}^{\nu-1}$. As per LogUp, $\{\alpha_i\}_{i=0}^{\mu-1} \subseteq \{\tau_j\}_{j=0}^{\nu-1}$ (where $\boldsymbol{\alpha}$ and $\boldsymbol{\tau}$ are both viewed as sets) if and only if there is a list of multiplicities $\boldsymbol{o} = (o_j)_{j=0}^{\nu-1}$ such that the below identity for set inclusion holds:

$$\sum_{i=0}^{\mu-1} \frac{1}{X - \alpha_i} = \sum_{j=0}^{\nu-1} \frac{o_j}{X - \tau_j}.$$

By Schwartz-Zippel Lemma, we can check this polynomial identity by evaluating it at a random point $X = c$. Here, c can be the random message from the verifier after receiving the commitment $\overline{Q}$ to $\boldsymbol{q} = ((\alpha_i)_{i=0}^{\mu-1}, (o_j)_{j=0}^{\nu-1})$ from the prover. Thanks to Fiat-Shamir transform [48], we can eliminate the interaction and compute $c := \rho(\overline{Q})$ instead. Consequently, the $\mathcal{F}^{\mathsf{lookup}}$ circuit needs to enforce 1) $\sum_{i=0}^{\mu-1} \frac{1}{c-\alpha_i} = \sum_{j=0}^{\nu-1} \frac{o_j}{c-\tau_j}$, i.e., the set inclusion identity holds, and 2) $c = \rho(\overline{Q})$, i.e., c is honestly computed.

Check 1) can be done by collecting the queries $(\alpha_i)_{i=0}^{\mu-1}$ from the execution of $\mathcal{F}$ circuit and asking the prover to feed $(o_j)_{j=0}^{\nu-1}$ and c as hints to $\mathcal{F}^{\mathsf{lookup}}$.

However, check 2) is more tricky. A naive yet problematic way is to let the prover feed $\overline{Q}$ as a hint and check if its digest is c. Nevertheless, this approach fails to guarantee soundness. Note that an honest $\overline{Q}$ should be the commitment to the queries made by $\mathcal{F}$ in the i-th step of IVC, which should thus be a part of the incoming instance $\mathbb{u}_{i+1}$, but the circuit is unable to verify this. In fact, $\mathbb{u}_{i+1}$ will be calculated *after* the current execution of $\mathcal{F}^{\mathsf{lookup}}$ and is unknown to $\mathcal{F}^{\mathsf{lookup}}$ *during* its execution.

To address this issue, we mark c as a public output and defer the check to the next step. More specifically, $\mathbb{u}_{i+1}.\boldsymbol{x}$, the statement of $\mathcal{F}^{\mathsf{aug}}$ in the i-th step, now contains $\varrho(\mathbb{U}_{i+1}, i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1})$ and c. With $\mathbb{u}_{i+1}.\overline{Q}$ and $\mathbb{u}_{i+1}.\boldsymbol{x}$, $\mathcal{F}^{\mathsf{lookup}}$ in the $(i+1)$-th step can now check the honesty of c from the i-th step by comparing it with $\rho(\mathbb{u}_{i+1}.\overline{Q})$. Analogously, $\mathcal{F}^{\mathsf{lookup}}$ in the i-th step becomes responsible for ensuring c from the $(i-1)$-th step is derived from $\mathbb{u}_i.\overline{Q}$.

Circuit 2: $\mathcal{F}^{\mathsf{lookup}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \overline{T}, \mathsf{aux}_i)$

```
Witness: $i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \overline{T}, \mathsf{aux}_i$
Statement: $h, c$
Constant: $(\tau_j)_{j=0}^{\nu-1}$
$\boldsymbol{z}_{i+1} := \mathcal{F}(\boldsymbol{z}_i; \mathsf{aux}_i)$    ▷ Let $(\alpha_i)_{i=0}^{\mu-1}$ be queries made by $\mathcal{F}$
Check $\mathbb{u}_i$:
    enforce $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \rho(\mathbb{u}_i.\overline{Q}))$
$\mathbb{U}_{i+1} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_i, \mathbb{u}_i, \overline{T})$
Check lookup queries:
    $\boldsymbol{o} \leftarrow \mathsf{Hint}(\boldsymbol{\alpha})$
    $c \leftarrow \mathsf{Hint}(\boldsymbol{\alpha}, \boldsymbol{o})$
    enforce $\sum_{i=0}^{\mu-1} \frac{1}{c-\alpha_i} = \sum_{j=0}^{\nu-1} \frac{o_j}{c-\tau_j}$
$h := \varrho((i = 0)\ ?\ \mathbb{U}_\perp : \mathbb{U}_{i+1}, i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1})$
return $h, c$
```

With $\mathcal{F}^{\mathsf{lookup}}$ as the augmented step circuit, we construct $\mathsf{IVC}^{\mathsf{lookup}}$ (Algorithm 3) accordingly, also by leveraging the folding-to-IVC compiler described above.

Formally, the prover takes as input the previous proof $\pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$, folds $(\mathbb{U}_i, \mathbb{W}_i)$ into $(\mathbb{u}_i, \mathbb{w}_i)$ to obtain $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1})$, and runs the $\mathcal{F}^{\mathsf{lookup}}$ circuit. When asked for hint $\boldsymbol{o}$ w.r.t. $\boldsymbol{\alpha}$, the prover computes o_j as the number of occurrences of table entry τ_j in $\boldsymbol{\alpha}$ for all $j \in [0, \nu-1]$. When asked for hint c w.r.t. $\boldsymbol{\alpha}, \boldsymbol{o}$, the prover computes $\overline{Q} \leftarrow \mathsf{CM}.\mathcal{C}(\mathsf{ck}, \boldsymbol{q})$ and $c := \rho(\overline{Q})$. Finally, the incoming instance-witness pair $(\mathbb{u}_{i+1}, \mathbb{w}_{i+1})$ corresponding to the i-th execution of $\mathcal{F}^{\mathsf{aug}}$ is constructed, and the updated proof $\pi_{i+1} = ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbb{u}_{i+1}, \mathbb{w}_{i+1}))$ is returned.

The verification of an IVC proof $\pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$ simply checks the digest h and challenge c in $\mathbb{u}_i.\boldsymbol{x}$ and ensures that $\mathbb{w}_i, \mathbb{W}_i$ satisfy $\mathbb{u}_i, \mathbb{U}_i$, respectively.

By applying this paradigm to Nova and relaxed R1CS [24], we obtain a folding scheme LouaFS and an IVC LouaIVC. We additionally utilize CycleFold [49] to improve circuit efficiency. Specifically, a CycleFold circuit $\mathcal{F}^{\mathsf{cf}}$ is constructed on $\mathbb{H}$ to handle group operations in $\mathsf{LouaFS}.\mathcal{V}$ in a native way, while the augmented step circuit on $\mathbb{G}$ only needs to perform field operations in $\mathsf{LouaFS}.\mathcal{V}$. As a trade-off, $\mathcal{F}^{\mathsf{aug}}$ becomes responsible for enforcing the correct folding of CycleFold instances $\mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}$ using $\mathsf{LouaFS}.\mathcal{V}$. Since $\mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}$ are over $\mathbb{H}$, the group operations in $\mathsf{NIFS}.\mathcal{V}$ can be handled natively by $\mathcal{F}^{\mathsf{aug}}$ over $\mathbb{G}$.

Comparison with other IVC with lookup. Several folding-based IVC schemes [26], [27], [50], [51] also support lookup arguments. In HyperNova [26], the authors build

Algorithm 3: $\mathsf{IVC}^{\mathsf{lookup}}$

```
Fn $\mathsf{IVC}^{\mathsf{lookup}}.\mathcal{G}(1^\lambda)$:
    return $\mathsf{pp} \leftarrow \mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{G}(1^\lambda)$
Fn $\mathsf{IVC}^{\mathsf{lookup}}.\mathcal{K}(\mathsf{pp}, \mathcal{F})$:
    Build $\mathcal{F}^{\mathsf{lookup}}$ for $\mathcal{F}$
    return $(\mathsf{pk}, \mathsf{vk}) := \mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{K}(\mathsf{pp}, \mathcal{F}^{\mathsf{lookup}})$
Fn $\mathsf{IVC}^{\mathsf{lookup}}.\mathcal{P}(\mathsf{pk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \mathsf{aux}_i, \pi_i)$:
    Parse $((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i)) := \pi_i$
    $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}, \overline{T}) := \mathsf{NIFS}^{\mathsf{lookup}}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$
    $(h, c) \leftarrow \mathcal{F}^{\mathsf{lookup}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \overline{T}, \mathsf{aux}_i)$
    Construct $\mathbb{u}_{i+1}, \mathbb{w}_{i+1}$ from the execution of $\mathcal{F}^{\mathsf{lookup}}$
    return $\pi_{i+1} := ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbb{u}_{i+1}, \mathbb{w}_{i+1}))$
Fn $\mathsf{IVC}^{\mathsf{lookup}}.\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i)$:
    Parse $((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i)) := \pi_i$
    assert $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \rho(\mathbb{u}_i.\overline{Q}))$
    assert $\mathbb{w}_i$ is a satisfying incoming witness to $\mathbb{u}_i$
    assert $\mathbb{W}_i$ is a satisfying running witness to $\mathbb{U}_i$
    return 1
```

nlookup upon the sum-check protocol. Similar to our paradigm, Protostar [27] and its subsequent work [50] utilizes LogUp as well. Very recently, NeutronNova [51] integrates Lasso [32] with folding.

For the use cases where the lookup table size ν is large, [26], [27], [50] provide more efficient solutions. However, our paradigm is optimal if the number of queries μ is much larger than ν, which is the case for our video encoding and editing circuits. In fact, the complexity of our paradigm is on par with existing schemes, but it further has minimal constant factor. For instance, our prover only needs to additionally commit to $\mu + 2\nu$ values, which is also the case for Protostar, while NeutronNova requires $3\mu+3\nu$. Moreover, we only add one additional commitment $\overline{Q}$ to folding instances when integrating lookup arguments, whereas Protostar and NeutronNova introduce two.

As a general and flexible approach, our paradigm supports any folding schemes and constraint systems. In comparison, the technique in NeutronNova leads to a dedicated folding scheme, and it is unclear how Protostar's solution can be combined with folding schemes that are not based on special-sound protocols [27].

4.2. Paradigm 2: Commit-and-Prove Decider

We introduce a decider Decider that compresses the final IVC proof π_k into a succinct zero-knowledge proof ϖ via a zkSNARK for the relation R^{IVC}. Given statement $\boldsymbol{x} = (k, \boldsymbol{z}_0, \boldsymbol{z}_k)$ and witness $\boldsymbol{w} = \pi_k$, $R^{\mathsf{IVC}}(\boldsymbol{x}, \boldsymbol{w}) = 1$ if and only if $\mathsf{IVC}.\mathcal{V}(\mathsf{vk}, (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k) = 1$.

Existing deciders. There are two existing methods for building Decider upon zkSNARKs [24], [35].

In Nova [24], the authors construct a dedicated Polynomial IOP [52] for relaxed R1CS and compile it into a zkSNARK for R^{IVC} using a polynomial commitment scheme (PCS). Two choices of the PCS are presented: a Pedersen-based PCS with Bulletproofs [53] as the IPA, and a two-tiered PCS (e.g., Dory-PC [54]) with Dory-IPA [54]. For $\mathcal{F}^{\mathsf{aug}}$ with n constraints, the former achieves $O(\log n)$ proof size and $O(n)$ verification time, while the latter makes both proof size and verification time logarithmic in n.

`sonobe` [35] instead expresses R^{IVC} as a circuit and prove its satisfiability with Groth16 [46], yielding constant proof size and verifier time. Nevertheless, `sonobe`'s decider only supports compressing proofs that use KZG commitment [55], where $\mathsf{IVC}.\mathcal{P}$ in each step needs to interpolate the polynomial from the input vector, resulting in an $O(n \log n)$ prover due to number-theoretic transforms (NTT).

Our construction. Our goal is to design a decider that improves both approaches. Specifically, it should produce constant-size proofs that can be verified in constant time w.r.t. n, while keeping the time of $\mathsf{IVC}.\mathcal{P}$ linear in n. To this end, we also express R^{IVC} as a circuit $\mathcal{F}^{\mathsf{Decider}}$ and prove its satisfiability, similar to `sonobe`. However, recall that R^{IVC} checks a portion of $\mathbb{W}_k$ (e.g., $\boldsymbol{q}$ in $\mathsf{NIFS}^{\mathsf{lookup}}$) against the commitments in $\mathbb{U}_k$ (e.g., $\overline{Q}$). We neither perform this check directly in-circuit (which is costly due to non-native elliptic curve operations), nor convert $\mathbb{W}_k$ to a polynomial and verify its evaluation against the polynomial commitments in $\mathbb{U}_k$ (which requires interpolation during the conversion).

Our approach leverages CP-SNARKs [34], which allow one to demonstrate that a subset of the witnesses in a SNARK indeed corresponds to a commitment, without running $\mathsf{CM}.\mathcal{V}$ in-circuit. Thus, the constraints for commitment verification are completely eliminated, whereas vector-to-polynomial conversion is also unnecessary because CP-SNARKs do not require polynomial commitments. Concretely, we can use Pedersen commitment [56] as CM by choosing LegoGro16 [34] as ZKCP, which establishes a bridge between Groth16 and Pedersen commitments. As a result, the prover time in each iteration of incremental proof generation is linear, and both the final compressed proof size and the verifier time are constant.

In addition, we adopt the trick in [24], [35] to further save computation in $\mathcal{F}^{\mathsf{Decider}}$, where $\mathsf{Decider}.\mathcal{P}$ is required to run $\mathsf{NIFS}.\mathcal{P}$ once more to absorb $(\mathbb{u}_k, \mathbb{w}_k)$ into $(\mathbb{U}_k, \mathbb{W}_k)$. Consequently, $\mathcal{F}^{\mathsf{Decider}}$ only needs to check the output $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1})$ instead of both inputs.

We present the general decider algorithm Decider in Algorithm 4. As discussed above, $\mathsf{Decider}.\mathcal{P}$ first runs $\mathsf{NIFS}.\mathcal{P}$ to fold $(\mathbb{u}_k, \mathbb{w}_k)$ into $(\mathbb{U}_k, \mathbb{W}_k)$, and then generates a proof ϖ that $\mathcal{F}^{\mathsf{Decider}}$ is satisfiable and that the commitments in $\mathbb{U}_{k+1}$ are valid with $\mathsf{ZKCP}.\mathcal{P}$. $\mathsf{Decider}.\mathcal{V}$ folds $\mathbb{u}_k$ into $\mathbb{U}_k$ as well, checks the public inputs in $\mathbb{u}_k.\boldsymbol{x}$, and verifies the proof ϖ and the commitments in $\mathbb{U}_{k+1}$ using $\mathsf{ZKCP}.\mathcal{V}$.

It is worth noting that, due to Groth16, $\mathsf{Decider}.\mathcal{P}$ has $O(n' \log n')$ complexity, where n' is the number of constraints in $\mathcal{F}^{\mathsf{Decider}}$ and is linear in n. But we stress that it is a one-time cost at the end of multiple steps of $\mathsf{IVC}.\mathcal{P}$, and it thus can be relatively cheap in practice.

Loua's decider LouaDecider can be constructed with LouaFS and LegoGro16, where the associated circuit $\mathcal{F}^{\mathsf{Decider}}$ needs to check the primary instance-witness pair $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1})$ and the CycleFold instance-witness pair

Algorithm 4: Decider

```
Fn Decider.$\mathcal{K}(1^\lambda, (\mathsf{ck}, \mathcal{F}^{\mathsf{Decider}}))$:
    return $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{ZKCP}.\mathcal{K}(1^\lambda, \mathsf{ck}, \mathcal{F}^{\mathsf{Decider}})$
Fn Decider.$\mathcal{P}((\mathsf{pk}, \mathsf{pk}_\Phi), (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k)$:
    Parse $((\mathbb{U}_k, \mathbb{W}_k), (\mathbb{u}_k, \mathbb{w}_k)) := \pi_k$
    $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1}, \overline{T}) := \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}_\Phi, (\mathbb{U}_k, \mathbb{W}_k), (\mathbb{u}_k, \mathbb{w}_k))$
    $\boldsymbol{c} :=$ (commitments in $\mathbb{U}_{k+1}$),
      $\boldsymbol{x} :=$ (other components of $\mathbb{U}_{k+1}$)
    $\boldsymbol{v} :=$ (committed values in $\mathbb{W}_{k+1}$),
      $\boldsymbol{\omega} :=$ (other components of $\mathbb{W}_{k+1}$)
    $\varpi \leftarrow \mathsf{ZKCP}.\mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{v}, \boldsymbol{\omega})$
    return $(\varpi, \mathbb{U}_k, \mathbb{u}_k, \overline{T})$
Fn Decider.$\mathcal{V}((\mathsf{vk}, \mathsf{vk}_\Phi), (k, \boldsymbol{z}_0, \boldsymbol{z}_k), (\varpi, \mathbb{U}_k, \mathbb{u}_k, \overline{T}))$:
    $\mathbb{U}_{k+1} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_k, \mathbb{u}_k, \overline{T})$
    assert $\mathbb{u}_k.\boldsymbol{x} = (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, \boldsymbol{z}_k), \rho(\mathbb{u}_k.\overline{Q}))$
    $\boldsymbol{c} :=$ (commitments in $\mathbb{U}_{k+1}$),
      $\boldsymbol{x} :=$ (other components of $\mathbb{U}_{k+1}$)
    return $\mathsf{ZKCP}.\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{c}, \varpi)$
```

$(\mathbb{U}_k^{\mathsf{cf}}, \mathbb{W}_k^{\mathsf{cf}})$. Since $\mathbb{U}_k^{\mathsf{cf}}$ and $\mathbb{W}_k^{\mathsf{cf}}$ are over the secondary curve $\mathbb{H}$, the commitment verification becomes native operations for circuits over $\mathbb{G}$. However, the satisfiability check $\boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v} \equiv u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e} \pmod{q}$ requires non-native field operations. We utilize tricks that eliminate in-circuit modulo operations, thereby reducing the cost of this check by 16x compared with the approach that naively utilizes [57].

5. The Eva Protocol

In this section, we introduce the construction of Eva, our proof of video authenticity based on IVC. In Section 5.1, we elaborate on the construction of our IVC step circuit $\mathcal{F}^{\mathsf{Eva}}$. Then, in Section 5.2, we build upon $\mathcal{F}^{\mathsf{Eva}}$ the full construction of Eva and discuss its security.

5.1. Building the Step Circuit

Recall that in a proof of video authenticity, $\mathcal{P}$ aims to convince $\mathcal{V}$ that the processed video stream ζ is honestly edited and encoded from some original video $\boldsymbol{V}$ whose signature σ is valid with respect to the public key vk_Σ. Due to the nature of video processing algorithms, we can view the editing and encoding operation as a sequence of subprocedures on each macroblock of the video. Thus, we first construct editing and encoding gadgets (small circuits that perform some specific operations) for a single macroblock. Then, we discuss how to combine both gadgets into $\mathcal{F}^{\mathsf{Eva}}$. Finally, we complete $\mathcal{F}^{\mathsf{Eva}}$ by adding the checks for the validity of signature σ.

Editing gadget. On input a macroblock $\boldsymbol{X}$, the editing gadget $\mathcal{F}^{\Delta}$ transforms $\boldsymbol{X}$ according to the editing parameter param^{Δ} and returns the edited macroblock $\boldsymbol{X}'$. Here, $\mathcal{F}^{\Delta}$ can be the in-circuit representation of arbitrary editing operation Δ. As examples, we construct several editing gadgets in Appendix A.2, which are for color manipulations (grayscale conversion, brightness adjustment, color inversion), spatial operations (masking, cropping), and temporal operations (cutting).

Encoding gadget. Naively, we can construct the encoding gadget $\mathcal{F}^{\mathcal{E}}$ by translating the encoder algorithm $\mathcal{E}$ to an arithmetic circuit. However, this would require a prohibitive number of constraints, because $\mathcal{E}$ involves complex operations such as motion estimation, entropy coding, etc., and the encoding of a macroblock may further depend on other macroblocks in the current or neighboring frame.

To address these challenges, we make extensive use of verifier's knowledge. Although video codecs are generally lossy, the decoder can still accurately extract from ζ some information that appears in the encoding process as well. In fact, the prediction macroblock $\boldsymbol{P}$ decoded by $\mathcal{D}$ is identical to the original prediction macroblock computed by $\mathcal{E}$, which is also the case for the quantized coefficients $\boldsymbol{Z}$.

Thus, we save the prover's cost by treating $\boldsymbol{P}$ and $\boldsymbol{Z}$ as public inputs, which can be recovered by $\mathcal{V}$. Now, to prove the honest encoding of a macroblock $\boldsymbol{X}$ with encoding parameters $\mathsf{param}^{\mathcal{E}}$, $\mathcal{F}^{\mathcal{E}}$ no longer runs the entire $\mathcal{E}$. Instead, $\mathcal{F}^{\mathcal{E}}$ only has to enforce the honest execution of differing, transform, and quantization, while it becomes unnecessary to prove prediction and entropy coding. This process is illustrated in Gadget 5. Here, the gadget $\mathcal{F}^{\mathsf{Diff}}$ for residual macroblock computation simply returns $\boldsymbol{R} := \boldsymbol{X} - \boldsymbol{P}$, while details of $\mathcal{F}^{\mathsf{Trans}}$ and $\mathcal{F}^{\mathsf{Quant}}$ are elaborated in Appendix A.1.

Gadget 5: $\mathcal{F}^{\mathcal{E}}(\boldsymbol{X}, \boldsymbol{P}, \mathsf{param}^{\mathcal{E}})$

```
$\boldsymbol{R} := \mathcal{F}^{\mathsf{Diff}}(\boldsymbol{X}, \boldsymbol{P})$                ▷ Compute residual macroblock
$\boldsymbol{Y} := \mathcal{F}^{\mathsf{Trans}}(\boldsymbol{R})$             ▷ Compute transformed coefficients
$\boldsymbol{Z} \leftarrow \mathcal{F}^{\mathsf{Quant}}(\boldsymbol{Y}, \mathsf{param}^{\mathcal{E}})$  ▷ Compute quantized coefficients
return $\boldsymbol{Z}$
```

Proof of Editing and Encoding. Now we integrate $\mathcal{F}^{\Delta}$ and $\mathcal{F}^{\mathcal{E}}$ into $\mathcal{F}^{\mathsf{Eva}}$. Since both gadgets extensively use bitwise operations, we fill the lookup table $\boldsymbol{\tau}$ with 2^8 entries in $\mathbb{Z}_{2^8}$ to maximize the efficiency. Then, for a macroblock $\boldsymbol{X}$, $\mathcal{F}^{\mathsf{Eva}}$ runs $\mathcal{F}^{\Delta}$ on $\boldsymbol{X}$ to obtain the edited macroblock $\boldsymbol{X}'$ and invokes $\mathcal{F}^{\mathcal{E}}$ on $\boldsymbol{X}'$ to get the quantized coefficients $\boldsymbol{Z}$.

We further extend $\mathcal{F}^{\mathsf{Eva}}$ to handle b macroblocks $(\boldsymbol{X}_j)_{j=0}^{b-1}$ in a batch, where each $\boldsymbol{X}_j$ is associated with public inputs $\boldsymbol{P}_j$ and $\boldsymbol{Z}_j$. With a reasonably large b, we can amortize the constraints for $\mathsf{NIFS}.\mathcal{V}$ in the augmented circuit $\mathcal{F}^{\mathsf{aug}}$, thereby enabling a more efficient IVC prover.

Nevertheless, the naive combination of $\mathcal{F}^{\Delta}$ and $\mathcal{F}^{\mathcal{E}}$ is suboptimal. Recall that in IVC, the circuit $\mathcal{F}^{\mathsf{aug}}$ computes $\varrho(\mathbb{U}_i, \cdot)$ and $\varrho(\mathbb{U}_{i+1}, \cdot)$, where $\mathbb{U}_i.\boldsymbol{x}$ and $\mathbb{U}_{i+1}.\boldsymbol{x}$ contain all the public inputs to the step circuit, which are $(\boldsymbol{P}_j)_{j=0}^{b-1}$ and $(\boldsymbol{Z}_j)_{j=0}^{b-1}$ in our case. Thus, the circuit needs to hash these data twice, which becomes expensive when b is large.

In fact, it is possible to avoid treating $(\boldsymbol{P}_j)_{j=0}^{b-1}$ and $(\boldsymbol{Z}_j)_{j=0}^{b-1}$ as public inputs while enjoying the shared information between encoder and decoder. Instead, we only treat them as witnesses, and the public input is now their digest $\hbar$. More specifically, in the i-th step of IVC, we absorb $(\boldsymbol{P}_{bi+j})_{j=0}^{b-1}$ and $(\boldsymbol{Z}_{bi+j})_{j=0}^{b-1}$ into $\hbar_i$ via H, thereby obtaining the next state $\hbar_{i+1}$. In this way, the prover only needs to compute the digest of $\boldsymbol{P}$ and $\boldsymbol{Z}$ once per step in IVC.

Proof of Valid Signature. Due to the hash-and-sign paradigm, the signature σ is actually for the digest of $\boldsymbol{V}$ and meta. By regarding the digest computation of $\boldsymbol{V}$ as an iterative invocation of H on each macroblock in the video, we can extend $\mathcal{F}^{\mathsf{Eva}}$ by hashing the original macroblock $\boldsymbol{X}$ as well. On the other hand, the hash of meta and the execution of $\mathsf{Sig}.\mathcal{V}$ are deferred to the end of IVC, as we will see in Section 5.2.

Now, the i-th state of IVC $\boldsymbol{z}_i$ not only contains the digest $\hbar_i$ of prediction macroblocks and quantized coefficients, but it also records h_i, the hash of macroblocks in the original video. In each step of $\mathcal{F}^{\mathsf{Eva}}$, the digest h_{i+1} is derived by absorbing the incoming macroblocks $(\boldsymbol{X}_{bi+j})_{j=0}^{b-1}$ into h_i.

Furthermore, to increase parallelism, we compute the digests h_{i+1} and $\hbar_{i+1}$ in two steps: 1) calculate the partial digests $h'_{bi+j} := \mathsf{H}(\boldsymbol{X}_{bi+j})$ and $\hbar'_{bi+j} := \mathsf{H}(\boldsymbol{P}_{bi+j}, \boldsymbol{Z}_{bi+j}, \mathsf{param}_{bi+j})$ for all $j \in [0, b-1]$, and 2) derive the final digests h_{i+1} and $\hbar_{i+1}$ by hashing the partial digests $(h'_{bi+j})_{j=0}^{b-1}$ and $(\hbar'_{bi+j})_{j=0}^{b-1}$.

The final construction of $\mathcal{F}^{\mathsf{Eva}}$ is given in Circuit 6.

Circuit 6: $\mathcal{F}^{\mathsf{Eva}}$

```
Witness: $\boldsymbol{z}_i, (\boldsymbol{X}_{bi+j})_{j=0}^{b-1}, (\mathsf{param}_{bi+j})_{j=0}^{b-1}$
$(h_i, \hbar_i) := \boldsymbol{z}_i$
for $j \in [0, b-1]$ do
    $\boldsymbol{X}'_{bi+j} \leftarrow \mathcal{F}^{\Delta}(\boldsymbol{X}_{bi+j}, \mathsf{param}^{\Delta}_{bi+j})$
    $\boldsymbol{P}_{bi+j} \leftarrow \mathsf{Hint}(\boldsymbol{X}'_{bi+j})$
    $\boldsymbol{Z}_{bi+j} \leftarrow \mathcal{F}^{\mathcal{E}}(\boldsymbol{X}'_{bi+j}, \boldsymbol{P}_{bi+j}, \mathsf{param}^{\mathcal{E}}_{bi+j})$
    $h'_{bi+j} := \mathsf{H}(\boldsymbol{X}_{bi+j})$
    $\hbar'_{bi+j} := \mathsf{H}(\boldsymbol{P}_{bi+j}, \boldsymbol{Z}_{bi+j}, \mathsf{param}_{bi+j})$
    if $\Delta = \Delta_{\mathsf{remove}}$ then
        $\hbar'_{bi+j} := \mathsf{param}^{\mathsf{remove}}_{bi+j}\ ?\ \mathsf{param}^{\mathsf{remove}}_{bi+j} : \hbar'_{bi+j}$
$h_{i+1} := \mathsf{H}(h_i, (h'_{bi+j})_{j=0}^{b-1})$
$\hbar_{i+1} := \mathsf{H}(\hbar_i, (\hbar'_{bi+j})_{j=0}^{b-1})$
if $\Delta = \Delta_{\mathsf{remove}}$ then
    $\hbar_{i+1} := (\bigwedge_{j \in [0,b-1]} \mathsf{param}^{\mathsf{remove}}_{bi+j})\ ?\ \hbar_i : \hbar_{i+1}$
return $\boldsymbol{z}_{i+1} := (h_{i+1}, \hbar_{i+1})$
```

In addition to the points discussed above, we take extra care to handle the possible removal of macroblocks due to the cropping or cutting operations ($\mathcal{F}^{\Delta_{\mathsf{remove}}}$ in Appendix A.2). For a removed macroblock $\boldsymbol{X}'$, $\mathcal{F}^{\mathcal{E}}$ and subsequent operations should not be performed, since $\boldsymbol{X}'$ is no longer encoded by $\mathcal{E}$.

This introduces different control flows depending on a dynamic parameter $\mathsf{param}^{\mathsf{remove}}$, resulting in a non-uniform circuit that is not directly supported by our IVC. A common technique to avoid this non-uniformity is to run all possible control flows, and then select among the results based on the dynamic branching condition: 1) Perform $\mathcal{F}^{\mathcal{E}}$ and H to derive $\hbar'$, as if $\boldsymbol{X}'$ is not removed. We can use dummy values for $\boldsymbol{X}'$, $\boldsymbol{P}$, $\boldsymbol{Z}$, and $\mathsf{param}^{\mathcal{E}}$ if they do not exist. 2) Compute $\hbar'$ without $\boldsymbol{P}$, $\boldsymbol{Z}$, and $\mathsf{param}^{\mathcal{E}}$, *i.e.*,, $\hbar' := \mathsf{H}(\mathsf{param}^{\mathsf{remove}})$. We further get rid of the hash and set $\hbar' := \mathsf{param}^{\mathsf{remove}}$, as the parameter only has a single bit. Later, we select between 1) and 2) based on the branching condition $\mathsf{param}^{\mathsf{remove}}$.

Nevertheless, this approach is still deficient. Recall that $\hbar$ is a public input computed by both $\mathcal{P}$ and $\mathcal{V}$. Thus, for a very large original footage $\boldsymbol{V}$, even the cropped (or cut) video ζ is very small, $\mathcal{V}$ still needs to compute the hash of dummy values for the non-existent $\boldsymbol{P}$ and $\boldsymbol{Z}$. In fact, $\mathcal{V}$'s costs are the same as if nothing is removed.

To save verification cost, one may consider running all control flows in-circuit when computing $\hbar_{i+1}$, *i.e.*, computing $\mathsf{H}(\hbar_i, S)$ for all $S \in 2^{(\hbar'_{bi+j})_{j=0}^{b-1}}$, where $2^{(\hbar'_{bi+j})_{j=0}^{b-1}}$ is the power set of $(\hbar'_{bi+j})_{j=0}^{b-1}$, and then the correct result can be selected. It is straightforward to see the downside of this approach: it significantly increases the prover's complexity.

We take a hybrid approach by reducing the number of branches to 2, depending on if *all* macroblocks in a batch of size b are discarded. If this is the case, the circuit selects the previous digest $\hbar_i$ as the next digest $\hbar_{i+1}$. Otherwise, the circuit selects $\mathsf{H}(\hbar_i, (\hbar'_{bi+j})_{j=0}^{b-1})$ as $\hbar_{i+1}$. As a result, $\mathcal{P}$ only needs to additionally handle 3 constraints while making it unnecessary for $\mathcal{V}$ to hash all dummy values. In fact, what $\mathcal{V}$ computes now is the hash of $\boldsymbol{P}$ and $\boldsymbol{Z}$ for a cropped (or cut) video whose size is padded to a multiple of the batch size b, which is pretty close to the actual size of ζ.

5.2. Final Protocol

Having built an IVC scheme based on our variant of Nova with support for lookup arguments in Section 4 and the IVC step circuit in Section 5.1, we now present Eva, a succinct, efficient, and secure proof of video authenticity.

In $\mathcal{K}_\Pi$, the trusted party instantiates $(\mathsf{pk}_\Pi, \mathsf{vk}_\Pi)$ with the proving and verification keys for the IVC scheme and the corresponding decider, and in $\mathcal{K}_\Sigma$, $(\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma)$ is obtained by invoking $\mathsf{Sig}.\mathcal{K}$, where Sig is instantiated with Schnorr signature [38]. Then, given the signing key sk_Σ, the recorder $\mathcal{R}$ computes the signature σ on the video $\boldsymbol{V}$ and its metadata meta as $\sigma \leftarrow \mathsf{Sig}.\mathcal{S}(\mathsf{sk}_\Sigma, \mathsf{H}(\boldsymbol{V}, \mathsf{meta}))$.

Next, we dive into the details of our prover $\mathcal{P}$ and verifier $\mathcal{V}$. $\mathcal{P}$ aims to convince $\mathcal{V}$ of the satisfiability of $\mathcal{F}^{\mathsf{Eva}}$. To this end, $\mathcal{P}$ first instantiates the lookup table $\boldsymbol{\tau}$ with 2^8 entries $\{0, \ldots, 255\}$. Then $\mathcal{P}$ prepares the inputs to $\mathcal{F}^{\mathsf{Eva}}$ by transforming $\boldsymbol{V}$ into $\boldsymbol{V}'$ via Δ, and using $\mathcal{E}$ to encode $\boldsymbol{V}'$, during which the quantized coefficients and prediction macroblocks are extracted. Next, $\mathcal{P}$ incrementally proves the satisfiability of $\mathcal{F}^{\mathsf{Eva}}$ using our Loua-based IVC. After $k = (M/16 \times N/16 \times L)/b$ steps, the final IVC state becomes $\boldsymbol{z}_k = (h_k, \hbar_k)$, where h_k is the digest of $(\boldsymbol{X}_i)_{i=0}^{bk-1}$, and $\hbar_k$ is the digest of $(\boldsymbol{P}_i)_{i=0}^{bk-1}$, $(\boldsymbol{Z}_i)_{i=0}^{bk-1}$, and $(\mathsf{param}_i)_{i=0}^{bk-1}$. Finally, $\mathcal{P}$ compresses the IVC proof with a decider based on ZKCP and returns the compressed zero-knowledge proof as well as the video stream ζ. These data are sent to $\mathcal{V}$, together with the metadata meta and editing and encoding parameters param.

As mentioned in Section 5.1, it is still left to compute $\mathsf{H}(h_k, \mathsf{meta})$ and run $\mathsf{Sig}.\mathcal{V}$ on the digest. Since $\mathsf{Decider}.\mathcal{V}$ takes the final state as public inputs, we can give $\mathcal{V}$ the hash

h_k and ask $\mathcal{V}$ to handle the rest of verification. However, this approach is suboptimal because of the weak security guarantee: h_k leaks information about the original video $\boldsymbol{V}$, leading to compromise of the zero-knowledge property.

To achieve full zero-knowledge, we exploit the flexibility of the decider circuit $\mathcal{F}^{\mathsf{Decider}}$ and hide h_k from $\mathcal{V}$. More specifically, we 1) verify σ on $\mathsf{H}(h_k, \mathsf{meta})$ under vk_Σ in $\mathcal{F}^{\mathsf{Decider}}$, and 2) move the computations related to $\mathbbm{u}_k.\boldsymbol{x}$ in $\mathsf{Decider}.\mathcal{V}$ to $\mathcal{F}^{\mathsf{Decider}}$, as h_k is required to derive the first component of $\mathbbm{u}_k.\boldsymbol{x}$, *i.e.*, $\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, \boldsymbol{z}_k)$.

In our adapted decider circuit $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$, the statement $\mathbb{U}'_k$ and $\mathbbm{u}'_k$ now no longer include $\boldsymbol{x}$. Instead, the prover provides h_k and $\boldsymbol{x}_k$ as witnesses, and the circuit reconstructs $\mathbb{U}_k$ by merging $\mathbb{U}'_k$ with $\boldsymbol{x}_k$, and $\mathbbm{u}_k$ by merging $\mathbbm{u}'_k$ with $(\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}^{\mathsf{cf}}_k, k), \rho(\mathbbm{u}_k.\overline{Q}))$. Then, the circuit computes $\mathbb{U}^{\mathbb{F}}_{k+1}$ using the field-only operation $\mathsf{NIFS}.\mathcal{V}^{\mathbb{F}}$, and finally, checks $\mathbb{W}_{k+1}$ against $\mathbb{U}^{\mathbb{F}}_{k+1}$. The final construction of $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ is given in Circuit 7.

Circuit 7: $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$

```
**Witness:** $h_k, \sigma, \boldsymbol{x}_k, \mathbb{W}_{k+1}, \mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k$
**Statement:** $\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T}$
**Constant:** $\mathsf{CS}^{\mathsf{aug}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}), \mathsf{CS}^{\mathsf{cf}} = (\boldsymbol{A}^{\mathsf{cf}}, \boldsymbol{B}^{\mathsf{cf}}, \boldsymbol{C}^{\mathsf{cf}}), \mathsf{ck}^{\mathsf{cf}}$
**enforce** $\mathsf{Sig}.\mathcal{V}(\mathsf{vk}_\Sigma, \sigma, \mathsf{H}(h_k, \mathsf{meta}))$        ▷ Verify $\sigma$
Reconstruct $\mathbb{U}_k$ and $\mathbbm{u}_k$:
    $\mathbb{U}_k := \mathbb{U}'_k$
    $\mathbb{U}_k.\boldsymbol{x} := \boldsymbol{x}_k$
    $\mathbbm{u}_k := \mathbbm{u}'_k$
    $\mathbbm{u}_k.\boldsymbol{x} := (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}^{\mathsf{cf}}_k, k), \rho(\mathbbm{u}_k.\overline{Q}))$
**enforce** $r = \rho(\mathbb{U}_k, \mathbbm{u}_k, \overline{T})$        ▷ Check $r$
$\mathbb{U}^{\mathbb{F}}_{k+1} := \mathsf{NIFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}, \mathbb{U}^{\mathbb{F}}_k, \mathbbm{u}^{\mathbb{F}}_k, r)$        ▷ Compute $\mathbb{U}^{\mathbb{F}}_{k+1}$
Check $\mathbb{W}_{k+1}$ against $\mathbb{U}^{\mathbb{F}}_{k+1}$:
    Parse $(u, \boldsymbol{x}) := \mathbb{U}^{\mathbb{F}}_{k+1}, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}_{k+1}$
    $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
    **enforce** $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = u \cdot \boldsymbol{C}\boldsymbol{v} + \boldsymbol{e}$
Check $\mathbb{W}^{\mathsf{cf}}_k$ against $\mathbb{U}^{\mathsf{cf}}_k$:
    Parse $(u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}) := \mathbb{U}^{\mathsf{cf}}_k, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}^{\mathsf{cf}}_k$
    $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
    **enforce** $\boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v} \equiv u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e} \pmod q$
    **enforce** $\boldsymbol{q} = \varnothing \wedge \overline{Q} = \overline{0}$
    **enforce** $\mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{w}, \overline{W})$
    **enforce** $\mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{e}, \overline{E})$
```

To verify the proof, $\mathcal{V}$ checks if the metadata meta and parameters param are acceptable. Similar to $\mathcal{P}$, $\mathcal{V}$ runs the decoding algorithm $\mathcal{D}$ on ζ to obtain $(\boldsymbol{P}_i)_{i=0}^{bk-1}$ and $(\boldsymbol{Z}_i)_{i=0}^{bk-1}$. After that, $\mathcal{V}$ computes $\hbar_k$ by hashing $(\boldsymbol{P}_i)_{i=0}^{bk-1}$, $(\boldsymbol{Z}_i)_{i=0}^{bk-1}$, and $(\mathsf{param}_i)_{i=0}^{bk-1}$. It is also $\mathcal{V}$'s task to check the commitments in $\mathbb{U}^{\mathbb{G}}_k, \mathbbm{u}^{\mathbb{G}}_k$ and $\mathbb{U}^{\mathbb{G}}_{k+1}$, which are not included in the decider circuit $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ due to the complexity of non-native group operations. With the randomness r and the cross term commitment $\overline{T}$, $\mathcal{V}$ derives $\mathbb{U}^{\mathbb{G}}_{k+1}$ by calling $\mathsf{NIFS}.\mathcal{V}^{\mathbb{G}}$ on $\mathbb{U}^{\mathbb{G}}_k, \mathbbm{u}^{\mathbb{G}}_k$. The commitments $\overline{Q}, \overline{W}, \overline{E}$ in $\mathbb{U}^{\mathbb{G}}_{k+1}$ are linked to the in-circuit witnesses $\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}$ in $\mathbb{W}_{k+1}$ via ZKCP. Note that $\mathcal{V}$ cannot learn h_k from $r := \rho(\mathbb{U}_k, \mathbbm{u}_k, \overline{T})$, since $\mathbb{U}_k.\boldsymbol{x}$, the random linear combination of all previous public inputs, is also kept secret. Finally, by running $\mathsf{ZKCP}.\mathcal{V}$, the verifier can check the authenticity of the video.

We summarize the Eva protocol in Algorithm 8.

Algorithm 8: Eva

```
**Fn** $\mathsf{Eva}.\mathcal{K}_\Sigma(1^\lambda)$**:**
    **return** $(\mathsf{sk}_\Sigma, \mathsf{vk}_\Sigma) \leftarrow \mathsf{Sig}.\mathcal{K}(1^\lambda)$
**Fn** $\mathsf{Eva}.\mathcal{K}_\Pi(1^\lambda)$**:**
    $\mathsf{pp} \leftarrow \mathsf{IVC}.\mathcal{G}(1^\lambda)$        ▷ pp contains ck
    $(\mathsf{pk}_\Phi, \mathsf{vk}_\Phi) := \mathsf{IVC}.\mathcal{K}(\mathsf{pp}, \mathcal{F}^{\mathsf{Eva}})$
    $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{ZKCP}.\mathcal{K}(1^\lambda, \mathsf{ck}, \mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}})$
    **return** $(\mathsf{pk}_\Pi := (\mathsf{pk}, \mathsf{pk}_\Phi), \mathsf{vk}_\Pi := (\mathsf{vk}, \mathsf{vk}_\Phi))$
**Fn** $\mathsf{Eva}.\mathcal{R}(\mathsf{sk}_\Sigma, \boldsymbol{V}, \mathsf{meta})$**:**
    **return** $\sigma \leftarrow \mathsf{Sig}.\mathcal{S}(\mathsf{sk}_\Sigma, \mathsf{H}(\boldsymbol{V}, \mathsf{meta}))$
**Fn** $\mathsf{Eva}.\mathcal{P}(\mathsf{pk}_\Pi, \mathsf{vk}_\Sigma, \boldsymbol{V}, \mathsf{meta}, \mathsf{param}, \sigma)$**:**
    $\boldsymbol{V}' := \Delta(\boldsymbol{V}, (\mathsf{param}^{\Delta}_i)_{i=0}^{bk-1})$
    Encode $\boldsymbol{V}'$ and extract $(\boldsymbol{P}_i)_{i=0}^{bk-1}, (\boldsymbol{Z}_i)_{i=0}^{bk-1}$:
        $\zeta := \mathcal{E}(\boldsymbol{V}', (\mathsf{param}^{\mathcal{E}}_i)_{i=0}^{bk-1})$
    $\boldsymbol{z}_0 := (0, 0),\ \pi_0 := ((\mathbb{U}_\perp, \mathbb{W}_\perp), (\mathbb{U}_\perp, \mathbb{W}_\perp), (\mathbb{U}^{\mathsf{cf}}_\perp, \mathbb{W}^{\mathsf{cf}}_\perp))$
    **for** $j \in [0, k)$ **do**
        $\mathsf{aux}_j := (\boldsymbol{X}_i, \boldsymbol{P}_i, \boldsymbol{Z}_i, \mathsf{param}_i)_{i=bj}^{bj+b-1}$
        $\pi_{j+1} \leftarrow \mathsf{IVC}.\mathcal{P}(\mathsf{pk}_\Phi, (j, \boldsymbol{z}_0, \boldsymbol{z}_j), \mathsf{aux}_j, \pi_j)$
        $\boldsymbol{z}_{j+1} := \mathcal{F}(\boldsymbol{z}_j; \mathsf{aux}_j)$
    Parse $(h_k, \hbar_k) := \boldsymbol{z}_k$,
      $((\mathbb{U}_k, \mathbb{W}_k), (\mathbbm{u}_k, \mathbbm{w}_k), (\mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k)) := \pi_k$
    $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1}, \overline{T}) := \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}_\Phi, (\mathbb{U}_k, \mathbb{W}_k), (\mathbbm{u}_k, \mathbbm{w}_k))$
    $r := \rho(\mathbb{U}_k, \mathbbm{u}_k, \overline{T})$
    $\boldsymbol{x} := (\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T}), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$
    $\boldsymbol{v} := (\mathbb{W}_{k+1}), \boldsymbol{\omega} := (h_k, \sigma, \mathbb{U}_k.\boldsymbol{x}, \mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k)$
    $\varpi \leftarrow \mathsf{ZKCP}.\mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{v}, \boldsymbol{\omega})$
    **return** $\zeta, \pi := (\varpi, \mathbb{U}'_k, \mathbbm{u}'_k, \overline{T}, r)$
**Fn** $\mathsf{Eva}.\mathcal{V}(\mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \zeta, \mathsf{meta}, \mathsf{param}, \pi)$**:**
    Parse $(\varpi, \mathbb{U}'_k, \mathbbm{u}'_k, \overline{T}, r) := \pi$
    Decode $\zeta$ and extract $(\boldsymbol{P}_i)_{i=0}^{bk-1}, (\boldsymbol{Z}_i)_{i=0}^{bk-1}$:
        $\widetilde{\boldsymbol{V}} := \mathcal{D}(\zeta, (\mathsf{param}^{\mathcal{E}}_i)_{i=0}^{bk-1})$
    $\hbar_0 := 0$
    **for** $j \in [0, k)$ **do**
        $\hbar_{j+1} := \mathsf{H}(\hbar_j, (\mathsf{H}(\boldsymbol{P}_i, \boldsymbol{Z}_i, \mathsf{param}_i))_{i=bj}^{bj+b-1})$
    $\mathbb{U}^{\mathbb{G}}_{k+1} := \mathsf{NIFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}_\Phi, \mathbb{U}^{\mathbb{G}}_k, \mathbbm{u}^{\mathbb{G}}_k, r, \overline{T})$
    **assert** $\mathbbm{u}'_k.u = 1, \mathbbm{u}'_k.\overline{E} = \overline{0}$        ▷ Check $\mathbbm{u}'_k$
    $\boldsymbol{x} := (\mathsf{vk}_\Sigma, \mathsf{meta}, k, (0, 0), \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T}), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$
    **return** $\mathsf{ZKCP}.\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{c}, \varpi)$
```

5.3. Security

We formally capture the security properties of Eva in Theorem 1, whose proof is deferred to Appendix B.

Theorem 1. Eva *is a succinct and zero-knowledge proof of video authenticity.*

6. Implementation and Evaluation

We rely on the H.264 reference implementation `JM` [58] to encode and decode videos.We modify its source code and hook the encoding and decoding processes to extract the prediction macroblocks and quantized coefficients, as they are necessary for proof generation and verification.

Then we develop Eva in Rust over the BN254/Grumpkin half-pairing cycle of curves. In the implementation[1], we use the `arkworks` library [59] for algebraic operations and circuit constructions. We build Loua, our variant of Nova, by improving the implementation in `sonobe` [35]. We also provide an alternative implementation of LegoGro16. Unlike the original implementation [60], ours is more flexible and performant: it allows for shared witnesses $(\boldsymbol{v})_{i=0}^{\ell-1}$ with arbitrary length and supports increased parallelism. We further employ various optimizations in our implementation. GPU is used to accelerate commitment and cross term computation in our NIFS, where the former is powered by `icicle` [61], while the latter is implemented by us. H, ϱ and ρ are instantiated with Griffin [37], a circuit-friendlier hash function than Poseidon [62]. A large batch size b is chosen to amortize the overhead for NIFS.$\mathcal{V}$ in the augmented circuit of Eva.

To evaluate our implementation of Eva, we compile it with multi-threading and AVX2 enabled, and run it on a consumer-grade PC equipped with an Intel Core i9-12900K CPU (16 cores, 24 threads) with 64 GB of RAM and an NVIDIA GeForce RTX 3080 GPU with 12 GB of VRAM.

For testing purposes, we utilize two raw video files that are widely used for video codec benchmarking, as shown in Figure 3. The first video, "foreman.yuv," contains 256 frames of size 352×288, which is used to demonstrate Eva's capability to handle a variety of editing operations. We apply several editing operations to it, including grayscale conversion, brightness adjustment, color inversion, masking, cropping, and cutting, and we give the preview of the edited videos in Figure 4. In the second video "bunny.yuv," each frame is of size 1280×720. To showcase the scalability and practicality of Eva, we test two clips, one has $L = 1800$ frames (1 minute at 30 FPS), and the other has $L = 3600$ frames (2 minute at 30 FPS).

(a) foreman.yuv (b) bunny.yuv

Figure 3: Preview of original videos in the test dataset

Microbenchmarks. We fix the batch size $b = 256$ and study how the editing operation Δ affects the prover performance in Eva. Table 2 summarizes the number of constraints in the augmented step circuit $\mathcal{F}^{\mathsf{aug}}$ and the decider circuit $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$. It also reports the running time as well as the RAM usage of IVC.$\mathcal{P}$ and Decider.$\mathcal{P}$. The detailed analysis of the number of constraints in $\mathcal{F}^{\mathsf{aug}}$ for different Δ can be found in Table 3. Across different operations, the running time of IVC.$\mathcal{P}$ ranges from 145 to 206 ms, and the peak RAM usage varies from 7 to 11 GB. Decider.$\mathcal{P}$ takes much more time ($65 \sim 80$ s) and RAM ($40 \sim 50$ GB) to compress an IVC proof, but it only happens once at the end of Eva.$\mathcal{P}$.

End-to-end performance. The end-to-end performance of Eva is evaluated based on the running time of each algorithm. We test Eva with $b = 256$ on both videos in the dataset. For "foreman.yuv", we apply various editing operations to it, while for "bunny.yuv", we consider different lengths. The results are presented in Table 4.

In Eva.$\mathcal{P}$, it takes $57 \sim 82$ seconds to finish all IVC steps for "foreman.yuv". Regarding "bunny.yuv", the IVC proof generation in total necessitates 1.19 hours for the 1-minute clip and 2.39 hours for the 2-minute clip. An additional $65 \sim 80$ seconds is needed to make the proof fully succinct and zero-knowledge by running ZKCP.$\mathcal{P}$ in our decider, which produces a constant-size proof of 448 bytes. The entire proof generation process is completed within 60 GB of RAM, which is primarily due to the additional ZKCP.$\mathcal{P}$ step.

From the results, we conclude that the total IVC time scales linearly with the video size. Specifically, the IVC time for any video of size $M \times N \times L$ can be estimated by computing the number of IVC steps $k = (M/16 \times N/16 \times L)/b$ and scaling the single step time of IVC.$\mathcal{P}$ in Table 2. This is confirmed by the results for "foreman.yuv" and "bunny.yuv" under operation Δ_{id} (and is also applicable for any other Δ): for the former, it requires $k = (352/16 \times 288/16 \times 256)/256 = 396$ steps, and thus the estimated total time is 168.436 ms $\times\ 396 = 66.701$ s, which equals the actual time in Table 4; similarly, for the 1-minute and 2-minute clips of "bunny.yuv", we can estimate that the total IVC time is 4263.536 s and 8527.072 s respectively, both within a relative error of 1% compared to the actual time.

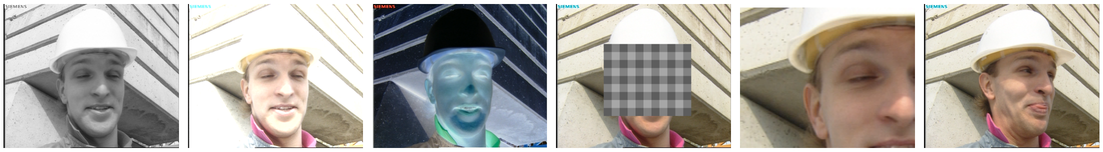
Another conclusion is that the ZKCP.$\mathcal{P}$ time is constant with respect to the number of IVC steps (and hence, the video size). Thus, although ZKCP.$\mathcal{P}$ constitutes a significant portion of the prover time for small videos, this one-time cost becomes less and less significant for larger videos.

The recorder $\mathcal{R}$'s time is dominated by the computation of Griffin hash, whose complexity is linear in the size of the *original* video $\boldsymbol{V}$. Similarly, H is also the bottleneck of $\mathcal{V}$, but it depends on the size of prediction macroblocks $\boldsymbol{P}$ and quantized coefficients $\boldsymbol{Z}$ of the *edited* video $\boldsymbol{V}'$. Thus, with Δ_{remove}, the verifier takes less time for computing H than other editing operations. In addition, $\mathcal{V}$ needs to validate the ZKCP proof by running ZKCP.$\mathcal{V}$, which always takes $2 \sim 3$ ms regardless of video size and editing operation.

Comparison with related work. Finally, we compare the performance of Eva with related work on image authentication based on zkSNARKs [18], [19], [20], [21], [22], focusing specifically on the prover time. PhotoProof [17] is not included in the comparison, as it only supports tiny images of size up to 128×128. Due to differences in image and video codecs, a common dataset cannot be used across all protocols. Consequently, the prover time is evaluated based on the number of pixels in the image or video.

For protocols that support arbitrary editing operations [19], [20], [21], [22], we select two representative operations for comparison: grayscale conversion and cropping. The target resolution for the cropping operation is set to

[1] The source code can be found here.



(a) Δ_{gray} (b) Δ_{bright} (c) Δ_{inv} (d) Δ_{mask} (e) Δ_{remove} (crop) (f) Δ_{remove} (cut)

Figure 4: Preview of videos edited from "foreman.yuv"

TABLE 2: Benchmarking results of $\mathsf{IVC}.\mathcal{P}$ and $\mathsf{Decider}.\mathcal{P}$ for different editing operation Δ with $b = 256$.

		Δ_{id}	Δ_{gray}	Δ_{bright}	Δ_{inv}	Δ_{mask}	Δ_{remove}
$\mathsf{IVC}.\mathcal{P}$	$\lvert\mathcal{F}^{\mathsf{aug}_{\mathsf{Eva}}}\rvert$	1429212	1101532	1887964	1429212	1802460	1528031
	Time (ms)	168.436	144.810	205.896	168.345	192.507	171.738
	Peak RAM (GB)	8.493	6.845	9.644	8.566	11.084	8.694
$\mathsf{Decider}.\mathcal{P}$	$\lvert\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}\rvert$	9539233	8916641	10391201	9539233	10187425	9736614
	Time (s)	69.233	65.017	73.499	69.274	80.391	70.630
	Peak RAM (GB)	44.226	39.560	48.601	45.705	52.766	43.797

TABLE 3: Breakdown of the number of R1CS constraints in $\mathcal{F}^{\mathsf{aug}}$ for $\mathcal{F}^{\mathsf{Eva}}$ with $b = 256$.

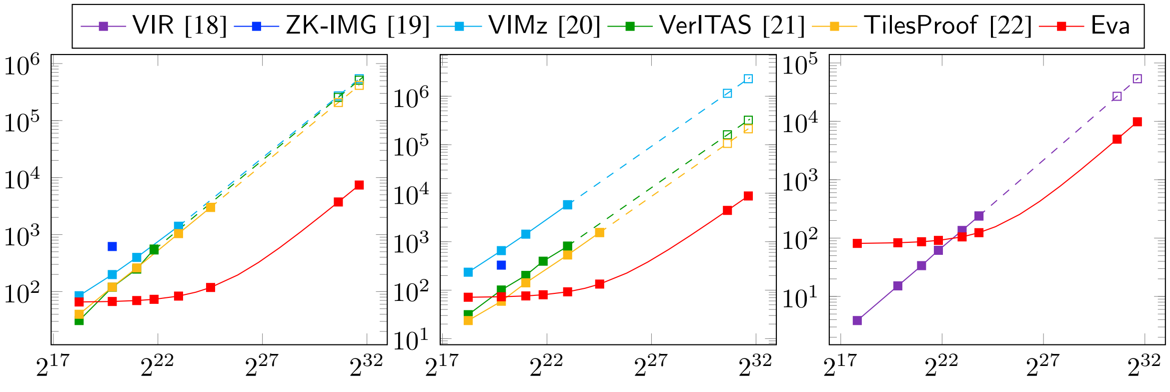
Subroutine				Editing Op.	Δ_{id}	Δ_{gray}	Δ_{bright}	Δ_{inv}	Δ_{mask}	Δ_{remove}	
$\mathcal{F}^{\mathsf{Eva}}$	Process $b = 256$ blocks			Create variables	42	42	42	42	426	43	$\times 256$
				$\mathcal{F}^{\Delta}$	0	0	1024	0	384	384	
		$\mathcal{F}^{\mathcal{E}}$		$\mathcal{F}^{\mathsf{Diff}}$	0						
				$\mathcal{F}^{\mathsf{Trans}}$	0						
			$\mathcal{F}^{\mathsf{Quant}}$	Y	1328						
				U	320	0	320	320	320	320	
				V	320	0	320	320	320	320	
				$\mathsf{H}(\boldsymbol{X})$	306						
				$\mathsf{H}(\boldsymbol{P}, \boldsymbol{Z}, \mathsf{param})$	612	612	612	612	918	613	
				$\mathsf{H}(h_i, \cdots)$	5508						
				$\mathsf{H}(\hbar_i, \cdots)$	5508	5508	5508	5508	5508	5511	
Subtotal					760584	596744	1022728	760584	1035528	859403	
Augmentation				Create variables	6865						
				Fold $\mathbb{u}_i^{\mathbb{F}}$ into $\mathbb{U}_i^{\mathbb{F}}$	13361						
				Fold $\mathbb{u}_i^{\mathsf{cf}}$ into $\mathbb{U}_i^{\mathsf{cf}}$	45108						
				Check lookup identity	594177	430337	790785	594177	692481	594177	
				Compute outputs	9117						
Subtotal					668628	504788	865236	668628	766932	668628	
Total					1429212	1101532	1887964	1429212	1802460	1528031	

TABLE 4: End-to-end performance of Eva with $b = 256$.

	L	Δ	$\mathcal{K}_{\Sigma}$ (μs)	$\mathcal{K}_{\Pi}$ (s)	$\mathcal{R}$		$\mathcal{P}$		$\mathcal{V}$	
					$\mathsf{H}(s)$	$\mathsf{Sig}.\mathcal{S}$(μs)	all steps of $\mathsf{IVC}.\mathcal{P}$ (s)	$\mathsf{ZKCP}.\mathcal{P}$ (s)	H (s)	$\mathsf{ZKCP}.\mathcal{V}$ (ms)
foreman	256	Δ_{id}	63.709	101.260	1.271	92.204	66.701	69.233	1.803	2.995
		Δ_{gray}	63.325	89.548	1.276	92.137	57.345	65.017	1.784	2.644
		Δ_{bright}	63.402	115.164	1.283	92.921	81.535	73.499	1.794	2.750
		Δ_{inv}	63.223	102.111	1.274	92.932	66.665	69.274	1.802	3.102
		Δ_{mask}	63.481	118.741	1.308	92.538	76.233	80.391	2.347	2.941
		Δ_{remove}	63.089	105.865	1.280	92.726	68.008	70.630	0.990 (crop) 0.883 (cut)	3.060
bunny	1800	Δ_{id}	63.250	101.913	81.713	92.488	4297.587	69.835	114.301	3.222
	3600	Δ_{id}	63.675	101.401	166.202	92.935	8617.108	70.277	229.339	3.084

640×480. Since the source code of ZK-IMG is not available, we rely on existing results for comparison. Specifically, we adopt the prover time for operations "RGB2YCbCr" and "Crop (HD → SD)" on HD (1280×720) images reported in [19, Table 4]. Note that ZK-IMG was evaluated on a powerful server with 64 CPU cores and 512 GB of RAM, suggesting that the prover would be slower on our machine. All remaining protocols are evaluated on our same machine. Figure 5a and Figure 5b show the results of prover time with a logarithmic scale on both axes.

In VIMz, VerITAS, TilesProof, and Eva, the prover time for both Δ_{gray} and Δ_{crop} increases nearly linearly with the number of pixels. We observe that Eva is generally faster than ZK-IMG, VIMz, VerITAS, and TilesProof, except at low resolutions (e.g., 640×480), where VerITAS and TilesProof outperform Eva due to our relatively long (but one-time) $\mathsf{ZKCP}.\mathcal{P}$ time. This also explains why the prover time for Eva does not appear as a straight line when the number of pixels is small. However, as the resolution increases, the ZKCP-based decider is no longer the dominant



(a) Grayscale conversion

(b) Cropping

(c) Masking

Figure 5: Comparison of prover time (y-axis, in seconds) across different protocols w.r.t. the number of pixels (x-axis).

factor in our prover, allowing the advantages of our efficient IVC.$\mathcal{P}$ to become apparent. In particular, for 4K resolution (3840 × 2160), Eva is 5.8 $\sim$ 92 times faster than VIMz, VerITAS and TilesProof. Further, we estimate that if we apply them to the 2-minute clip of "bunny.yuv" even without considering memory constraints and lossy encoding, their proof generation would be respectively 73 $\sim$ 262, 36 $\sim$ 69, and 24 $\sim$ 56 times longer than Eva, depending on Δ.

For VIR [18], their redaction operation is equivalent to our masking operation with black tiles as the mask. Thus, we compare Eva with VIR in terms of Δ_{mask} and present the results in Figure 5c. To ensure a fair comparison, we set the granularity of redaction (*i.e.*, the minimum size of black tiles) in VIR to 1×1, matching the granularity of our masking operation. For relatively small number of pixels, our prover takes longer than VIR due to the one-time cost of the decider. However, when the data size increases, the prover time of VIR increases more rapidly than ours, and Eva begin to outperform VIR for 4K and larger resolution. We also estimate that, even with unlimited RAM, VIR is 5.5x slower than Eva when proving the masking operation for the 2-minute "bunny.yuv".

7. Discussion

We further explore practical considerations for deploying Eva in real-world scenarios.

On-chain verification. It is possible to deploy Eva on blockchains to provide on-chain verification of video authenticity. More specifically, with $\hbar_k$ computed by the user, the smart contract can check if the proof π is valid. This is practical because π is on the BN254 curve, which is natively supported by Ethereum and its Layer 2 solutions. Also, thanks to our design of decider based on LegoGroth16 [34], π is very small and only require 2 pairings for verification. We estimate that verifying π on EVM requires $\sim$ 362000 gas, or equivalently $\sim$ 16 USD as of August 2024.

In comparison, although Dziembowski et al. claim that VIMz [20] supports on-chain verification, the concrete costs of their smart contracts are not provided in their paper, which turn out to be prohibitively high. In fact, they choose Spartan as the zkSNARK for decider and rely on the `solidity-verifier` library [63] for verifying Spartan proofs on EVM, requiring $\sim$ 200M gas or $\sim$ 9000 USD.

Implementation of the recorder. Note that in Eva, both $\mathcal{R}$ and $\mathcal{P}$ take the raw footage $\boldsymbol{V}$ as input. This implies that $\mathcal{R}$ should send to $\mathcal{P}$ the recorded video $\boldsymbol{V}$ *as is*, in an *uncompressed* or *losslessly encoded* manner.

In practice, $\mathcal{R}$ is usually resource-constrained and hence only allows lossily encoded videos. Thus, $\mathcal{P}$ needs to decode $\widetilde{\boldsymbol{V}}$ from the encoded video stream ζ before editing and proving. But due to information loss, $\widetilde{\boldsymbol{V}}$ is different from the original video $\boldsymbol{V}$ that $\mathcal{R}$ signs, leading to a mismatch between the signed video and the video to be proven.

To address this mismatch, an intuitive solution is to require $\mathcal{R}$ to sign ζ. Then, $\mathcal{P}$ needs to prove 1) $\mathsf{Sig}.\mathcal{V}(\mathsf{vk}_\Sigma, \zeta, \sigma)$, 2) honest editing and encoding on $\widetilde{\boldsymbol{V}}$, and additionally 3) $\widetilde{\boldsymbol{V}} = \mathcal{D}(\zeta)$ to connect 1) with 2). Nevertheless, this approach is impractical due to the complex decoding algorithm $\mathcal{D}$.

We adopt a more strategic approach, where $\mathcal{R}$ takes an additional step of decoding ζ and signs the decoded $\widetilde{\boldsymbol{V}}$ instead of ζ. This ensures that the video that $\mathcal{R}$ signs is exactly the one proven by $\mathcal{P}$, thereby eliminating the need for proving correct decoding. Here, $\mathcal{R}$ does not need to store the decoded $\widetilde{\boldsymbol{V}}$ for signature generation, as $\mathcal{R}$ can hash $\widetilde{\boldsymbol{V}}$ on-the-fly: $\mathcal{R}$ maintains a short digest as the accumulated hash, and once a new macroblock is populated by the decoder, $\mathcal{R}$ absorbs it into the accumulated digest, which can then be discarded.

Acknowledgments

This research was partially funded by the Engineering and Physical Sciences Research Council (EPSRC) under grant EP/R012598/1 and EP/S030867/1. The authors would like to thank Yihua Cheng for the discussion on related work based on detection, and the developers of `sonobe` for reviewing the pull request for optimizing non-native field operations in the decider circuit. Chengru Zhang was visiting UCL when working on this research.

References

[1] Dan Milmo. Youtube is major conduit of fake news, factcheckers say. https://www.theguardian.com/technology/2022/jan/12/youtube-is-major-conduit-of-fake-news-factcheckers-say.

[2] Emma Tucker. Tiktok's search engine repeatedly delivers misinformation to its majority-young user base, report says. https://edition.cnn.com/2022/09/18/business/tiktok-search-engine-misinformation/index.html.

[3] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[4] Zach Evans, CJ Carr, Josiah Taylor, Scott H Hawley, and Jordi Pons. Fast timing-conditioned latent audio diffusion. In *Forty-first International Conference on Machine Learning*, 2024.

[5] Dan Kondratyuk, Lijun Yu, Xiuye Gu, Jose Lezama, Jonathan Huang, Grant Schindler, Rachel Hornung, Vighnesh Birodkar, Jimmy Yan, Ming-Chang Chiu, et al. Videopoet: A large language model for zero-shot video generation. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Rashid Tahir, Brishna Batool, Hira Jamshed, Mahnoor Jameel, Mubashir Anwar, Faizan Ahmed, Muhammad Adeel Zaffar, and Muhammad Fareed Zaffar. Seeing is believing: Exploring perceptual differences in deepfake videos. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–16, 2021.

[7] Matthew Groh, Ziv Epstein, Chaz Firestone, and Rosalind Picard. Deepfake detection by human crowds, machines, and machine-informed crowds. *Proceedings of the National Academy of Sciences*, 119(1):e2110013119, 2022.

[8] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3207–3216, 2020.

[9] Alexandros Haliassos, Konstantinos Vougioukas, Stavros Petridis, and Maja Pantic. Lips don't lie: A generalisable and robust approach to face forgery detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5039–5049, 2021.

[10] Coalition for Content Provenance and Authenticity. *Content Credentials: C2PA Technical Specification*, 2.0 edition, 2024.

[11] Baris Coskun, Bulent Sankur, and Nasir Memon. Spatio–temporal transform based video hashing. *ieee Transactions on Multimedia*, 8(6):1190–1208, 2006.

[12] Fouad Khelifi and Ahmed Bouridane. Perceptual video hashing for content identification and authentication. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(1):50–67, 2017.

[13] Paarth Neekhara, Shehzeen Hussain, Xinqiao Zhang, Ke Huang, Julian McAuley, and Farinaz Koushanfar. Facesigns: Semi-fragile watermarks for media authentication. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2024.

[14] Iacopo Masi, Aditya Killekar, Royston Marian Mascarenhas, Shenoy Pratik Gurudatt, and Wael AbdAlmageed. Two-branch recurrent network for isolating deepfakes in videos. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 667–684. Springer, 2020.

[15] Shehzeen Hussain, Paarth Neekhara, Malhar Jere, Farinaz Koushanfar, and Julian McAuley. Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3348–3357, 2021.

[16] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar Jois, and Yinzhi Cao. Squint hard enough: Attacking perceptual hashing with adversarial machine learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 211–228, 2023.

[17] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 255–271. IEEE, 2016.

[18] Hankyung Ko, Ingeun Lee, Seunghwa Lee, Jihye Kim, and Hyunok Oh. Efficient verifiable image redacting based on zk-SNARKs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 213–226, 2021.

[19] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Zk-img: Attested images via zero-knowledge proofs to fight disinformation. *arXiv preprint arXiv:2211.04775*, 2022.

[20] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. VIMz: Verifiable image manipulation using folding-based zk-SNARKs. *Cryptology ePrint Archive*, 2024.

[21] Trisha Datta, Binyi Chen, and Dan Boneh. Veritas: Verifying image transformations at scale. *Cryptology ePrint Archive*, 2024.

[22] Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. Trust nobody: Privacy-preserving proofs for edited photos with your laptop. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 14–14. IEEE Computer Society, 2024.

[23] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 1–18. Springer, 2008.

[24] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.

[25] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.

[26] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. In *Annual International Cryptology Conference*, pages 345–379. Springer, 2024.

[27] Benedikt Bünz and Binyi Chen. Protostar: generic efficient accumulation/folding for special-sound protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 77–110. Springer, 2023.

[28] Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. *Cryptology ePrint Archive*, 2023.

[29] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 595–626. Springer, 2018.

[30] Ariel Gabizon and Zachary J Williamson. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive*, 2020.

[31] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3121–3134, 2022.

[32] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with Lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 180–209. Springer, 2024.

[33] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. *Cryptology ePrint Archive*, 2022.

[34] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2075–2092, 2019.

[35] Privacy & Scaling Explorations. sonobe: Experimental folding schemes library. https://github.com/privacy-scaling-explorations/sonobe, 2023.

[36] ITU Telecommunication Standardization Sector. *H.264: Advanced video coding for generic audiovisual services*, 14 edition, 2021.

[37] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. Horst meets fluid-spn: Griffin for zero-knowledge applications. In *Annual International Cryptology Conference*, pages 573–606. Springer, 2023.

[38] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.

[39] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010*, volume 10, pages 310–331, 2010.

[40] Zcash. The Halo2 zero-knowledge proving system. https://github.com/zcash/halo2, 2022.

[41] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*, pages 3–33. Springer, 2021.

[42] ITU Telecommunication Standardization Sector. *H.265: High efficiency video coding*, 9 edition, 2023.

[43] Alliance for Open Media. *AV1 Bitstream & Decoding Process Specification*, 1 edition, 2019.

[44] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 681–710. Springer, 2021.

[45] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. gnark is a fast zk-snark library that offers a high-level api to design circuits. https://github.com/Consensys/gnark, 2023.

[46] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

[47] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[48] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[49] Abhiram Kothapalli and Srinath Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. *Cryptology ePrint Archive*, 2023.

[50] Benedikt Bünz and Jessica Chen. Proofs for deep thought: Accumulation for large memories and deterministic computations. *Cryptology ePrint Archive*, 2024.

[51] Abhiram Kothapalli and Srinath Setty. Neutronnova: Folding everything that reduces to zero-check. *Cryptology ePrint Archive*, 2024.

[52] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 677–706. Springer, 2020.

[53] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.

[54] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.

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

[56] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[57] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xjsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 944–961. IEEE, 2018.

[58] MPEG & VCEG Joint Video Team. H.264/AVC JM reference software. https://vcgit.hhi.fraunhofer.de/jvet/JM, 2019.

[59] arkworks contributors. `arkworks` zksnark ecosystem. https://arkworks.rs, 2022.

[60] Lovesh Harchandani. legogro16: LegoGroth16 implementation on top of Zexe. https://github.com/lovesh/legogro16, 2023.

[61] Ingonyama. icicle: a GPU library for zero-knowledge acceleration. https://github.com/ingonyama-zk/icicle, 2023.

[62] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535, 2021.

[63] Lurk Lab. solidity-verifier: Solidity implementation of nova proving system verifier. https://github.com/lurk-lab/solidity-verifier, 2023.

[64] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-Knowledge Location Privacy via Accurate Floating-Point SNARKs . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 57–57, Los Alamitos, CA, USA, 2025. IEEE Computer Society.

[65] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 55–72. Springer, 2013.

Appendix A. Encoding and Editing Gadgets

A.1. Gadgets for Video Encoding

We provide efficient constructions of gadgets for transform and quantization operations compatible with H.264.

A.1.1. Transform. The transform operation in H.264 is based on 4×4 DCT (Discrete Cosine Transform), but for efficiency, it only involves integer operations, while the fractional part of the DCT coefficients is handled in the quantization step. With the core transform matrix $\boldsymbol{C}_4$, the transform on a 4×4 block $\boldsymbol{\Lambda}$ is computed as $\boldsymbol{\Gamma} := \boldsymbol{C}_4 \boldsymbol{\Lambda} \boldsymbol{C}_4^{\intercal}$. Hence, to construct the transform gadget $\mathcal{F}^{\mathsf{Trans}}$, we divide the color components $\boldsymbol{R}^{\mathsf{Y}}, \boldsymbol{R}^{\mathsf{Cb}}, \boldsymbol{R}^{\mathsf{Cr}}$ of a residual macroblock $\boldsymbol{R}$ into 4×4 blocks and apply the transform operation on each block. Since every entry $r_{i,j}$ in $\boldsymbol{R}$ is in $[-255, 255]$, the matrix multiplication can be natively performed in $\mathbb{F}_p$ without overflow.

After the core transform, the DC (i.e., the first) coefficients of all blocks from every color component are collected into a 4×4 matrix $\boldsymbol{B}^{\mathsf{Y}}$ and two 2×2 matrices $\boldsymbol{B}^{\mathsf{Cb}}, \boldsymbol{B}^{\mathsf{Cr}}$, while the AC (i.e., the remaining) coefficients are unchanged. These matrices are transformed again using Hadamard matrices $\boldsymbol{H}_4$ and $\boldsymbol{H}_2$, respectively. Finally, the transformed DC and AC coefficients $\boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}}$, $(\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}$ are returned. The entire in-circuit transform process is depicted in Gadget 9.

Gadget 9: $\mathcal{F}^{\mathsf{Trans}}(\boldsymbol{R})$

```
$\boldsymbol{C}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}, \boldsymbol{H}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}, \boldsymbol{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
for $k \in [0, 16)$ do
    $i := \lfloor k/4 \rfloor, j := k \bmod 4$
    $\boldsymbol{A}_k^{\mathsf{Y}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Y}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^{\intercal}$
    $b_{i,j}^{\mathsf{Y}} := a_{k,0,0}^{\mathsf{Y}}$
for $k \in [0, 4)$ do
    $i := \lfloor k/2 \rfloor, j := k \bmod 2$
    $\boldsymbol{A}_k^{\mathsf{Cb}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Cb}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^{\intercal}$
    $\boldsymbol{A}_k^{\mathsf{Cr}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Cr}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^{\intercal}$
    $b_{i,j}^{\mathsf{Cb}} := a_{k,0,0}^{\mathsf{Cb}}$
    $b_{i,j}^{\mathsf{Cr}} := a_{k,0,0}^{\mathsf{Cr}}$
$\boldsymbol{D}^{\mathsf{Y}} := \boldsymbol{H}_4 \boldsymbol{B}^{\mathsf{Y}} \boldsymbol{H}_4^{\intercal}$
$\boldsymbol{D}^{\mathsf{Cb}} := \boldsymbol{H}_2 \boldsymbol{B}^{\mathsf{Cb}} \boldsymbol{H}_2^{\intercal}$
$\boldsymbol{D}^{\mathsf{Cr}} := \boldsymbol{H}_2 \boldsymbol{B}^{\mathsf{Cr}} \boldsymbol{H}_2^{\intercal}$
$\boldsymbol{Y} := ((\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}, \boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}})$
return $\boldsymbol{Y}$
```

A.1.2. Quantization. The quantization step maps a coefficient v from the transform step to a quantized value u. This process is lossy, and how much information is preserved is controlled by the quantization parameter qp in H.264.

In general, the quantized coefficient is computed by $u := \lfloor v \times \psi / 2^{\delta} \rceil$, i.e., we first scale the transformed coefficient v by $\psi / 2^{\delta}$ and then round the result to the nearest integer. Here, ψ is the multiplication factor that takes the fractional part of the DCT coefficients into account, and δ is the number of bits to be right-shifted. In practice, $\lfloor v \times \psi / 2^{\delta} \rceil$ is done with approximate integer operations, which are more efficient in hardware. Specifically, we first compute the absolute value $\mathrm{abs}(u) := (\mathrm{abs}(v) \times \psi + \phi) \gg \delta$, where ϕ is an offset determined by qp, and u is then derived based on the sign $\mathrm{sign}(u) := \mathrm{sign}(v)$.

When dealing with scaling and rounding in-circuit, we leverage the efficient gadgets $\mathcal{F}^{\mathsf{SignAbs}}$ (Gadget 11) and $\mathcal{F}^{\gg}$ (Gadget 12) in [64] for computing absolute values and shifting operations. Both gadgets are built upon $\mathcal{F}^{\mathsf{EnforceBitLen}}$ (Gadget 10), which enforces that the input x has at most W bits by making queries to a lookup table. Such queries can be efficiently checked by our IVC with lookup arguments integrated. For details of the construction, please refer to [64].

With these gadgets, we give the construction of the $\mathcal{F}^{\mathsf{ScaleRound}}$ gadget for scaling and rounding an input coefficient v in-place in Gadget 13. Now we are finally ready to present the quantization gadget $\mathcal{F}^{\mathsf{Quant}}$. For an AC coefficient $a_{i,j}$, the multiplication factor is in the $(\mathsf{qp} \bmod 6)$-th row and the $p_{i,j}$-th column of the matrix of multiplication factors $\boldsymbol{\Psi}$, where $p_{i,j}$ is an element in another matrix $\boldsymbol{P}$. On the other hand, the multiplication factor for DC coefficients are always $\psi_{0,0}$. Then, for all AC and DC blocks, we apply the $\mathcal{F}^{\mathsf{ScaleRound}}$ gadget to quantize their coefficients with the corresponding parameters, except that the DC coefficients for luma are right shifted by 1 bit before quantization.

Gadget 10: $\mathcal{F}^{\mathsf{EnforceBitLen}}(x, W)$

```
$(x_i)_{i=0}^{W/\log\nu - 1} \leftarrow \mathsf{Hint}(x)$
enforce $\sum_{i=0}^{W/\log\nu - 1} 2^{i \log \nu} x_i = x$
$\boldsymbol{\alpha} := (\boldsymbol{\alpha}, (x_i)_{i=0}^{W/\log\nu - 1})$
```

Gadget 11: $\mathcal{F}^{\mathsf{SignAbs}}(x \in [-2^W + 1, 2^W - 1])$

```
$s \leftarrow \mathsf{Hint}(x)$
$y := s \,?\, x : -x$
enforce $s(1 - s) = 0$
$\mathcal{F}^{\mathsf{EnforceBitLen}}(y, W)$
return $s, y$
```

Gadget 12: $\mathcal{F}^{\gg}(x \in [0, 2^W - 1], \delta \in [U, V])$

```
$x' := x \cdot 2^{V - \delta}$
$q, r \leftarrow \mathsf{Hint}(x')$
enforce $x' = q \cdot 2^V + r$
$\mathcal{F}^{\mathsf{EnforceBitLen}}(q, W - U)$
$\mathcal{F}^{\mathsf{EnforceBitLen}}(r, V)$
```

Gadget 13: $\mathcal{F}^{\mathsf{ScaleRound}}(v, \psi, \phi, \delta)$

```
$(s, u) \leftarrow \mathcal{F}^{\mathsf{SignAbs}}(v)$
$t := \mathcal{F}^{\gg}(u \times \psi + \phi, \delta)$
$v := s \,?\, t : -t$
```

The entire quantization process is summarized in Gadget 14, with qp and mode included in $\mathsf{param}^{\mathcal{E}}$.

A.2. Gadgets for Video Editing

We showcase gadgets for video editing, including color manipulations (grayscale conversion, brightness adjustment, color inversion) and spatial and temporal operations (masking, cropping, cutting). Additionally, we explain how to perform complex editing operations that involve multiple macroblocks in-circuit.

A.2.1. Color Manipulations. Thanks to the use of the YCbCr color space, it is straightforward to perform common color manipulations for videos encoded in H.264 or in many other video codecs. In contrast, color operations on RGB often involve the conversion between color spaces, demanding for in-circuit fixed-point or floating-point computation.

For instance, when converting pixels in grayscale, we can simply keep the Y component unchanged while setting

```
Gadget 14: $\mathcal{F}^{\mathsf{Quant}}(\boldsymbol{Y}, \mathsf{param}^{\mathcal{E}})$
Parse $((\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}, \boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}}) := \boldsymbol{Y}$
Parse $(\mathsf{qp}, \mathsf{mode}, \cdot) := \mathsf{param}^{\mathcal{E}}$
$q := \lfloor \mathsf{qp}/6 \rfloor, r := \mathsf{qp} \bmod 6$
$f := ((\mathsf{mode} = \text{“intra”})\ ?\ 682 : 342) \times 2^{4+q}$
$\boldsymbol{\Psi} := \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix}, \boldsymbol{P} := \begin{bmatrix} 0 & 2 & 0 & 2 \\ 2 & 1 & 2 & 1 \\ 0 & 2 & 0 & 2 \\ 2 & 1 & 2 & 1 \end{bmatrix}$
for $\boldsymbol{A} \in ((\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4})$ do
    for $i \in [0,4), j \in [0,4)$ do
        $\mathcal{F}^{\mathsf{ScaleRound}}(a_{i,j}, \psi_{r,p_{i,j}}, f, 15+q)$
for $i \in [0,4), j \in [0,4)$ do
    $\mathcal{F}^{\mathsf{ScaleRound}}(\mathcal{F}^{\gg}(d_{i,j}^{\mathsf{Y}}, 1), \psi_{0,0}, 2f, 16+q)$
for $\boldsymbol{D} \in (\boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}})$ do
    for $i \in [0,2), j \in [0,2)$ do
        $\mathcal{F}^{\mathsf{ScaleRound}}(d_{i,j}, \psi_{0,0}, 2f, 16+q)$
$\boldsymbol{Z} := ((\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}, \boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}})$
return $\boldsymbol{Z}$
```

the chroma components to 128, because the luma component already represents the luminance in YCbCr. The corresponding gadget is depicted in Gadget 15.

```
Gadget 15: $\mathcal{F}^{\Delta_{\mathsf{gray}}}(\boldsymbol{X})$
Parse $(\boldsymbol{X}^{\mathsf{Y}}, \cdot) := \boldsymbol{X}$
return $\boldsymbol{X}' := (\boldsymbol{X}^{\mathsf{Y}}, \mathbf{128}, \mathbf{128})$
```

When adjusting the brightness, we only need to focus on the luma component, which is scaled by a factor $\mathsf{param}^{\mathsf{bright}}$ and clamped to $[0, 255]$, as shown in Gadget 16. We support 65536 levels of brightness adjustment, with $\mathsf{param}^{\mathsf{bright}} = \frac{\beta}{256} \in \{\frac{0}{256}, \frac{1}{256}, \ldots, \frac{65535}{256}\}$. For a luma component x^{Y}, the gadget first computes $\beta \times x^{\mathsf{Y}}$ and then right shifts the product by 8 bits. In order to clamp the product to $[0, 255]$, we again shift the result to the right by 8 bits. We return the original result if the remaining bits are all 0, otherwise 255.

```
Gadget 16: $\mathcal{F}^{\Delta_{\mathsf{bright}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{bright}} = \frac{\beta}{256})$
Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) := \boldsymbol{X}$
for $i \in [0,16), j \in [0,16)$ do
    $u := \mathcal{F}^{\gg}(x_{i,j}^{\mathsf{Y}} \times \beta, 8)$
    $v := \mathcal{F}^{\gg}(u, 8)$
    $x_{i,j}^{\mathsf{Y}} := (v = 0)\ ?\ u : 255$
return $\boldsymbol{X}' := (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

Gadget 17 inverts the color of a macroblock by subtracting all components in each pixel value from 255.

```
Gadget 17: $\mathcal{F}^{\Delta_{\mathsf{inv}}}(\boldsymbol{X})$
Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) := \boldsymbol{X}$
for $i \in [0,16), j \in [0,16)$ do
    $x_{i,j}^{\mathsf{Y}} := 255 - x_{i,j}^{\mathsf{Y}}$
for $i \in [0,8), j \in [0,8)$ do
    $x_{i,j}^{\mathsf{Cb}} := 255 - x_{i,j}^{\mathsf{Cb}}$
    $x_{i,j}^{\mathsf{Cr}} := 255 - x_{i,j}^{\mathsf{Cr}}$
return $\boldsymbol{X}' := (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

A.2.2. Spatial and Temporal Operations.

Now we present the gadgets for spatial and temporal operations.

To mask a macroblock $\boldsymbol{X}$ with an overlay $\boldsymbol{L}$, we additionally require a binary matrix $\boldsymbol{B}$, where each bit $b_{i,j}$ indicates if the pixel $x_{i,j}$ is replaced with the corresponding $l_{i,j}$. With $(\boldsymbol{B}, \boldsymbol{L})$ as the masking parameter $\mathsf{param}^{\mathsf{mask}}$, the masking gadget $\mathcal{F}^{\Delta_{\mathsf{mask}}}$ is given in Gadget 18. Note that different macroblocks may have different $\mathsf{param}^{\mathsf{mask}}$, which allows for arbitrary overlays with dynamic content and position (e.g., subtitles) without incurring additional costs.

```
Gadget 18: $\mathcal{F}^{\Delta_{\mathsf{mask}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{mask}} = (\boldsymbol{B}, \boldsymbol{L}))$
Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) := \boldsymbol{X}$
Parse $(\boldsymbol{B}^{\mathsf{Y}}, \boldsymbol{B}^{\mathsf{Cb}}, \boldsymbol{B}^{\mathsf{Cr}}) := \boldsymbol{B}$
Parse $(\boldsymbol{L}^{\mathsf{Y}}, \boldsymbol{L}^{\mathsf{Cb}}, \boldsymbol{L}^{\mathsf{Cr}}) := \boldsymbol{L}$
for $i \in [0,16), j \in [0,16)$ do
    $x_{i,j}^{\mathsf{Y}} := b_{i,j}^{\mathsf{Y}}\ ?\ l_{i,j}^{\mathsf{Y}} : x_{i,j}^{\mathsf{Y}}$
for $i \in [0,8), j \in [0,8)$ do
    $x_{i,j}^{\mathsf{Cb}} := b_{i,j}^{\mathsf{Cb}}\ ?\ l_{i,j}^{\mathsf{Cb}} : x_{i,j}^{\mathsf{Cb}}$
    $x_{i,j}^{\mathsf{Cr}} := b_{i,j}^{\mathsf{Cr}}\ ?\ l_{i,j}^{\mathsf{Cr}} : x_{i,j}^{\mathsf{Cr}}$
return $\boldsymbol{X}' := (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

Cropping and cutting both work similarly to each other, where the former removes data in the horizontal and vertical directions, while the latter removes data in the temporal direction. We unify both cases via the removal parameter $\mathsf{param}^{\mathsf{remove}}$, which consists of a boolean value b that indicates if the macroblocks needs to be removed. By specifying b according to the operation type, we can support both operations with the same gadget $\mathcal{F}^{\Delta_{\mathsf{remove}}}$. For instance, cropping requires $b = 1$ for macroblocks outside the cropped region, while for cutting, all macroblocks in removed frames have $b = 1$. The construction of $\mathcal{F}^{\Delta_{\mathsf{remove}}}$ is shown in Gadget 19, where $\perp$ is a dummy macroblock. Although the process seems straightforward, we omit an important detail in the description: how to handle $\perp$ is in fact non-trivial, and we defer the discussion to Section 5.1.

```
Gadget 19: $\mathcal{F}^{\Delta_{\mathsf{remove}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{remove}} = b)$
Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) := \boldsymbol{X}$
return $\boldsymbol{X}' := b\ ?\ (\perp, \perp, \perp) : (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

We also want to point out that while we require each macroblock to have its own $\mathsf{param}^{\mathsf{remove}}$, we can avoid linear communication complexity when transmitting the parameters from the prover $\mathcal{P}$ to the verifier $\mathcal{V}$. In fact, $\mathcal{P}$ can simply send the dimensions of the original video and the offset of the cropped or cut video with respect to the original one, and $\mathcal{V}$ can recover the parameters from these values.

A.2.3. More Complicated Operations. We discuss how to build gadgets for more complex editing operations that involve multiple macroblocks, such as rotation. While $\mathcal{F}^{\Delta}$ handles macroblocks one-by-one in our design, it still allows such advanced functionalities. To this end, we leverage vector commitment schemes [65] (e.g., Merkle trees), where one can commit to the entire vector of messages and later open the commitment to the message at a specific position.

Now, $\mathcal{F}^{\Delta}$ additionally takes as input the vector commitment to the original video $\boldsymbol{V}$. For an editing operation that *reads* both the current macroblock $\boldsymbol{X}_i$ and another macroblock $\boldsymbol{X}_j$, the prover can feed $\boldsymbol{X}_j$ to $\mathcal{F}^{\Delta}$ as a hint, and $\mathcal{F}^{\Delta}$ enforces that $\boldsymbol{X}_j$ is indeed the j-th macroblock in the video by checking the vector commitment against $\boldsymbol{X}_j$ and j. Similarly, we can also support operations that *update* macroblocks in different positions by including the vector commitment to the edited video $\boldsymbol{V}'$ as input. In this way, $\mathcal{F}^{\Delta}$ is able to access other macroblocks in the video. without affecting its macroblock-wise design.

Appendix B. Security of Eva

We prove Theorem 1 and show that Eva is *succinct*, *complete*, *knowledge sound*, and *zero-knowledge*.

Proof of succinctness. Eva satisfy succinctness because its proofs are of constant size. Specifically, the $\mathsf{LegoGro16}$ proof ϖ has 4 $\mathbb{G}$ elements and 1 $\hat{\mathbb{G}}$ element, the partial running instance $\mathbb{U}'_k$ has 3 $\mathbb{G}$ elements and 1 $\mathbb{F}_p$ element, the partial incoming instance $\mathbbm{u}'_k$ has 2 $\mathbb{G}$ elements, $\overline{T}$ is in $\mathbb{G}$, and r is in $\mathbb{F}_p$. In total, the proof π consists of 10 $\mathbb{G}$ elements, 1 $\hat{\mathbb{G}}$ element, and 2 $\mathbb{F}_p$ elements. □

Proof of completeness. We omit the proof of completeness for Eva, as it is straightforward to see from the design of our circuits and the completeness of IVC, NIFS, and ZKCP. □

Proof of knowledge soundness. We prove the knowledge soundness of Eva by constructing an efficient extractor Ext. Given public parameters $\mathsf{pk}_\Pi, \mathsf{vk}_\Pi, \mathsf{vk}_\Sigma$, the trapdoor td, and $\mathcal{A}$'s output $(\zeta, \mathsf{meta}, \mathsf{param}, \pi)$, we have $\mathcal{V}(\mathsf{vk}_\Pi, \mathsf{vk}_\Sigma, \zeta, \mathsf{meta}, \mathsf{param}, \pi) = 1$ by condition. Hence, $\mathsf{ZKCP}.\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{c}, \varpi) = 1$, for $\boldsymbol{x} := (\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T})$, $\boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$. With this condition, Ext works as below:

1) Invoke ZKCP's extractor on input $\boldsymbol{x}, \boldsymbol{c}, \varpi$. Except with negligible probability, Ext can obtain $\boldsymbol{v} := (\mathbb{W}_{k+1})$, $\boldsymbol{\omega} := (h_k, \sigma, \mathbb{U}_k.\boldsymbol{x}, \mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k)$, such that $(\boldsymbol{x}, \boldsymbol{c})$ and $(\boldsymbol{v}, \boldsymbol{\omega})$ satisfy $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$, and $\boldsymbol{v} = \mathbb{W}_{k+1}$ opens $\boldsymbol{c} = \mathbb{U}^{\mathbb{G}}_{k+1}$.
2) Reconstruct $\mathbb{U}_k$ from $\mathbb{U}'_k$ and $\mathbb{U}_k.\boldsymbol{x}$.
3) Reconstruct $\mathbbm{u}_k$ from $\mathbbm{u}'_k$ and $\mathbbm{u}_k.\boldsymbol{x} := (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}^{\mathsf{cf}}_k, k), \rho(\mathbbm{u}_k.\overline{Q}))$.
4) Line 4 of Circuit 7 enforces that $\mathbb{U}^{\mathbb{F}}_{k+1} := \mathsf{NIFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}, \mathbb{U}^{\mathbb{F}}_k, \mathbbm{u}^{\mathbb{F}}_k, r)$. Also, we have $\mathbb{U}^{\mathbb{G}}_{k+1} := \mathsf{NIFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}_\Phi, \mathbb{U}^{\mathbb{G}}_k, \mathbbm{u}^{\mathbb{G}}_k, r, \overline{T})$. Thus, $\mathbb{U}_{k+1} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_k, \mathbbm{u}_k, \overline{T})$. Moreover, Line 5 of Circuit 7 and the commit-and-prove relation w.r.t. $\boldsymbol{v} = \mathbb{W}_{k+1}$ and $\boldsymbol{c} = \mathbb{U}^{\mathbb{G}}_{k+1}$ imply that $\mathbb{W}_{k+1}$ and $\mathbb{U}_{k+1}$ satisfy $\mathsf{CS}^{\mathsf{aug}}$. Consequently, except with negligible probability, Ext can invoke the extractor of NIFS on input $\mathbb{U}_k, \mathbbm{u}_k, \mathbb{W}_{k+1}, \overline{T}$ and obtain $\mathbb{W}_k, \mathbbm{w}_k$ such that $(\mathbb{U}_k, \mathbb{W}_k)$ and $(\mathbbm{u}_k, \mathbbm{w}_k)$ satisfy $\mathsf{CS}^{\mathsf{aug}}$.
5) By the checks in Line 6 of Circuit 7, we can deduce that $\mathbb{U}^{\mathsf{cf}}_k$ and $\mathbb{W}^{\mathsf{cf}}_k$ satisfy $\mathsf{CS}^{\mathsf{cf}}$. At this point, Ext can recover $\pi_k := ((\mathbb{U}_k, \mathbb{W}_k), (\mathbbm{u}_k, \mathbbm{w}_k), (\mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k))$ such that all checks in $\mathsf{IVC}.\mathcal{V}(\mathsf{vk}, (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k) = 1$ pass. Consequently, except with negligible probability, Ext can invoke the extractor of IVC on input $\mathcal{F}^{\mathsf{aug}}, k, \boldsymbol{z}_0, \boldsymbol{z}_k, \pi_k$ and obtain the state and proof at $k-1$-th step.
6) Repeatedly invoke the extractor of IVC on the last state and proof, and obtain the previous state and proof, until reaching the initial step.
7) Parse the original video $\boldsymbol{V}$ from all the auxiliary states (aux_i) and return $\sigma, \boldsymbol{V}$.

By the satisfiability of $\mathcal{F}^{\mathsf{aug}}$, we can conclude that, with param, $(\boldsymbol{Z}_i)$ is the correct encoding of an video $\boldsymbol{V}'$ edited from the original video $\boldsymbol{V}$ whose digest is h_k. Also, by construction of $\mathsf{ZKCP}.\mathcal{V}$, ζ is the entropy coded bitstream of $(\boldsymbol{Z}_i)$. Furthermore, by Line 1 of Circuit 7, σ is a valid signature on $\mathsf{H}(h_k, \mathsf{meta})$. Thus, Ext successfully extracts $\boldsymbol{V}$ and σ such that $R^{\mathsf{VA}}((\zeta, \mathsf{meta}, \mathsf{param}, \mathsf{vk}_\Sigma), (\sigma, \boldsymbol{V})) = 1$, except with negligible probability. □

Proof of zero-knowledge. For zero-knowledge, we leverage the technique in [24, Appendix D] to construct a simulator Sim who can produce $\mathbb{U}_k, \mathbbm{u}_k$ that are indistinguishable from the outputs of the honest prover, if ϱ and CM are hiding.

First, Sim uniformly samples random values r_1, r_2, r_q, r_w, and initiates $\mathbb{U}_1, \mathbbm{u}_1$, where $\mathbb{U}_1 = \mathbb{U}_\perp$, $\mathbbm{u}_1.u = 1$, $\mathbbm{u}_1.\overline{Q} = \mathsf{CM}.\mathcal{C}(\mathsf{ck}, r_q)$, $\mathbbm{u}_1.\overline{W} = \mathsf{CM}.\mathcal{C}(\mathsf{ck}, r_w)$, $\mathbbm{u}_1.\overline{E} = \overline{0}$, $\mathbbm{u}_1.\boldsymbol{x} = (\varrho(r_1), \varrho(r_2), \rho(\mathbbm{u}_1.\overline{Q}))$. Here, $\mathbb{U}_1$, $\mathbbm{u}_1.u$, and $\mathbbm{u}_1.\overline{E}$ are equal to real ones. Also, since we assume ϱ and CM are hiding, $\mathbbm{u}_1.\overline{Q}$, $\mathbbm{u}_i.\overline{W}$, and $\mathbbm{u}_1.\boldsymbol{x}$ are indistinguishable from real ones.

Next, we show that for every i, if $\mathbb{U}_i$ and $\mathbbm{u}_i$ are indistinguishable from real ones, then Sim can generate $\mathbb{U}_{i+1}$ and $\mathbbm{u}_{i+1}$ that are also indistinguishable from real ones. To this end, Sim uniformly samples randomness r_1, r_2, r_q, r_w, r_t and computes $\overline{T} := \mathsf{CM}.\mathcal{C}(\mathsf{ck}, r_t)$, which is indistinguishable from real commitments since CM is hiding. Then, Sim computes $\mathbb{U}_{i+1} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_i, \mathbbm{u}_i, \overline{T})$. Further, $\mathbbm{u}_{i+1}$ is derived in the same way as the base case. In this way, both $\mathbb{U}_{i+1}$ and $\mathbbm{u}_{i+1}$ are indistinguishable from real instances.

After k steps, $\mathbb{U}_k, \mathbbm{u}_k$ are indistinguishable from the honestly generated ones. Again, Sim computes $\overline{T} := \mathsf{CM}.\mathcal{C}(\mathsf{ck}, r_t)$ for a random r_t and $r := \rho(\mathbb{U}_k, \mathbbm{u}_k, \overline{T})$, which are indistinguishable from real ones.

Sim then computes $\hbar_k$ by hashing the prediction macroblocks and quantized coefficients decoded from ζ, and derives $\mathbb{U}^{\mathbb{G}}_{k+1} := \mathsf{NIFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}_\Phi, \mathbb{U}^{\mathbb{G}}_k, \mathbbm{u}^{\mathbb{G}}_k, r, \overline{T})$.

Finally, Sim invokes the ZKCP simulator on input $\boldsymbol{x}$ and $\boldsymbol{c}$, where $\boldsymbol{x} := (\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T}), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$. The ZKCP simulator returns a simulated proof ϖ that is indistinguishable from the honestly generated ones, and the proof $\pi := (\varpi, \mathbb{U}'_k, \mathbbm{u}'_k, \overline{T}, r)$ that Sim returns is therefore also indistinguishable from the honest proofs. □

Appendix C.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper introduces Eva, a cryptographic scheme designed for authenticating the provenance of lossily-encoded and edited videos. Eva combines folding/accumulation-based incremental verifiable computation with novel mechanisms for handling public intermediate data to scale up to prove the correctness of edits and compression of the signed video.

C.2. Scientific Contributions

- Provides a valuable step forward in an Established Field

C.3. Reasons for Acceptance

1) Eva is the first cryptographic framework for authenticating the provenance of edited videos.
2) Eva develops and leverages novel insights about public data in the encoder/decoder algorithms of lossy-video-compression to reduce the size of circuits being proved.
3) Eva implements and evaluates their system on real-world video clips, and demonstrates that this their techniques are feasible (if not fully practical).