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

Chengru Zhang[1], Xiao Yang[2], David Oswald[2], Mark Ryan[2], and Philipp Jovanovic[3]

[1]The University of Hong Kong, Hong Kong
[2]University of Birmingham, UK
[3]University College London, UK

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 [67, 82]. This trend has been exacerbated recently by the rapid evolution of (generative) AI tools [77, 29, 53] 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 [81, 38] or automated methods [62, 43], 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 [22, 23, 50]. Among authentication-based approaches, the Coalition for Content Provenance and Authenticity (C2PA) standard [22] 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 [71, 23, 50] 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 information that is not intended for disclosure, such as the thumbnail of the original footage. Furthermore, many methods based on detection [81, 38, 62, 65, 43] and authentication [71, 23, 50] 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 [46, 76]. Cryptographic solutions have been proposed for *image* authentication [69, 52, 48, 26, 24, 25] 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. In particular, given a lossy-encoded video, 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 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) [83] constructed from *folding schemes* [60, 59, 57, 11, 27] 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* [8, 32, 84, 80] 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 [42], an efficient lookup argument [8], 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) [17], without introducing vector-to-polynomial conversions or evaluation arguments as in [60, 75].

By applying our paradigms to a popular folding scheme Nova [60], we introduce a variant Loua (named after **Lo**g**U**p and Nov**a**), which offers efficiency improvements over both the original Nova and its design in the `sonobe` folding library [75]. 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 [40] and employ a circuit-friendly hash function Griffin [35] and Schnorr signature [78], 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 $\sim$ 2.4 hours on a consumer-grade desktop. During IVC proof generation, the RAM usage is kept at a constant $\sim$ 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.

PhotoProof [69] is a pioneering work in this direction that uses *Proof-Carrying Data* (PCD) [21] to offer authenticity of edited images. Due to the high computational cost of proof generation, it only supports tiny images. In [52], Ko et al. propose VIR, which utilizes CP-SNARKs [17] to generate constant-sized 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` [85], a more efficient proof system, ZK-IMG [48] also has faster prover than PhotoProof while maintaining support for arbitrary editing operations.

More recent schemes include VIMz [26], VerITAS [24], and TilesProof [25]. 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

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 [69]	Image	Lossless	Arbitrary	$O(P^3 \log P)$ (< 18676 µs/px)	$O(P)$	$O(1)$ (2.67 KB)	128×128 $< \infty$
VIR [52]	Image	Lossless	Masking	$O(P \log P)$ (~ 16 µs/px)	$O(P)$	$O(1)$ (223 B)	3840×2160 $< \infty$
ZK-IMG [48]	Image	Lossless	Arbitrary	$O(P \log P)$ (> 355 µs/px	$O(P)$	$O(\log P)$ ($\sim$10 KB)	1280×720 $< \infty$
VIMz [26]	Image	Lossless	Arbitrary	$O(P)$ (~ 167 µs/px)	$O(N)$	$O(\log^2 N)$ ($\sim$10 KB)	3840×2160 $< \infty$
VerITAS [24]	Image	Lossless	Arbitrary	$O(P \log P)$ (~ 95 µs/px)	$O(P)$	$O(\log^2 P)$ ($\sim$100 KB)	6632×4976 $< \infty$
TilesProof [25]	Image	Lossless	Arbitrary	$O(P \log(P/T))$ (~ 62 µs/px)	$O(P/T)$	$O(T)$ ($\sim$10 KB)	6000×4000 $< \infty$
Eva (this work)	Video	Lossy (H.264) Lossless	Arbitrary	$O(P)$ (~ 2.6 µs/px)	$O(1)$	$O(1)$ (448 B)	$1280 \times 720 \times 3600$ ∞

[a] Asymptotic complexity is measured w.r.t. the number of pixels $P = MNL$ and the number of tiles T (required by [25]), 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).

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.

For details of performance evaluation, we refer the reader to Section 7. An extended review of related work can be found in Appendix A.

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

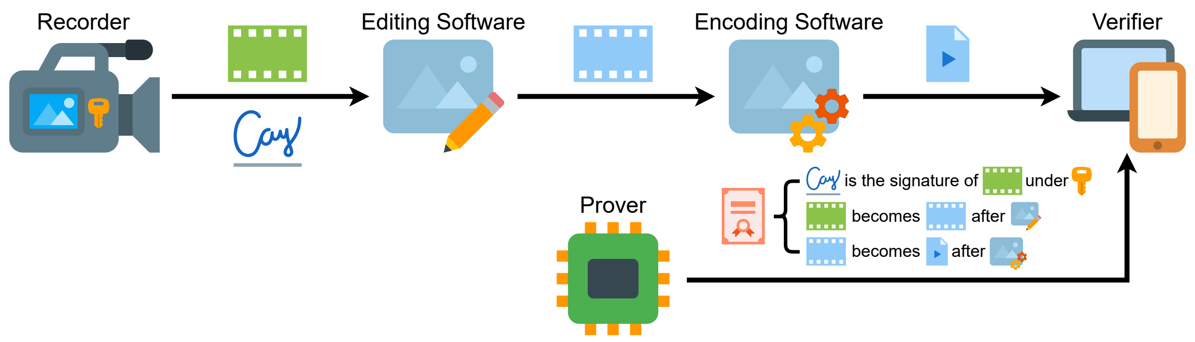


Figure 1: Overview of the Eva protocol. The recorder produces a raw footage and signs it with its embedded signing key. The footage is then edited and encoded into a video stream. The prover generates a proof attesting to the authenticity of the video. The verifier can check the proof of authenticity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

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

2.1 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 (whose unary representation is 1^λ), a negligible function in λ is denoted by $\varepsilon(\lambda)$.

Vectors and matrices are denoted by boldface italic lowercase (e.g., $\boldsymbol{x} = (x_0, x_1, \ldots))$ and uppercase letters (e.g., $\boldsymbol{X} = \begin{bmatrix} x_{0,0} & \cdots \\ \vdots & \ddots \end{bmatrix}$), respectively. $\boldsymbol{x}[i, j]$ is the 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. When it is clear from the context, we write $\boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z})$ to indicate that $\boldsymbol{x}$ is the concatenation of $\boldsymbol{y}$ and $\boldsymbol{z}$.

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. In this cycle, $\mathbb{F}_q$, the base field (*i.e.*, the field over which the curve is defined) of $\mathbb{G}$, is also the scalar field (*i.e.*, the prime field modulo the order of the curve) of $\mathbb{H}$; and $\mathbb{F}_p$, the scalar field of $\mathbb{G}$, is also the base field of $\mathbb{H}$. Further, $(\mathbb{G}, \widehat{\mathbb{G}}, \mathbb{G}_T)$ is a pairing-friendly group, *i.e.*, there is a bilinear map $e : \mathbb{G} \times \widehat{\mathbb{G}} \to \mathbb{G}_T$.

Algorithms are written in pseudocode, and we distinguish between the operations inside and outside an arithmetic circuit by using "Circuit" and "Algorithm" prefixes, respectively. Also, "Gadget" refers to a small circuit that performs a specific operation, which is often used as a building block in larger circuits. "$cond\,?\,x : y$" is a conditional expression that evaluates to x if the condition $cond$ is true, and y otherwise. The notation "**assert** $cond$" represents an operation that returns 0 if $cond$ is not satisfied and does nothing otherwise. Its in-circuit equivalent, "**enforce** $x = y$", adds a constraint to the constraint system to enforce equality between x and y. Additionally, *hints* refer to the non-deterministic advice [6] provided by the prover to the circuit.

2.2 Cryptographic Primitives

We rely on three collision-resistant hash functions H, ρ, and ϱ, an EUF-CMA secure signature scheme $\mathsf{Sig} = (\mathsf{Sig}.\mathcal{K}, \mathsf{Sig}.\mathcal{S}, \mathsf{Sig}.\mathcal{V})$ under chosen-message attack, and a commitment scheme $\mathsf{CM} = (\mathsf{CM}.\mathcal{K}, \mathsf{CM}.\mathcal{C}, \mathsf{CM}.\mathcal{V})$ that is binding and hiding, which we assume the reader is familiar with.

Here, we consider ρ as a random oracle in the random oracle model. Further, we define $\varrho(x; r) = \rho(x \,||\, r)$ 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 treat the randomness in ϱ, $\mathsf{CM}.\mathcal{C}$ and $\mathsf{CM}.\mathcal{V}$ as implicit and omit it from the notation.

2.3 SNARKs, CP-SNARKs, and Lookup Arguments

Consider a relation R with an associated NP-language L_R. For a *statement* $\boldsymbol{x}$, $\boldsymbol{x} \in L_R$ if there exists a *witness* $\boldsymbol{w}$ such that $\boldsymbol{x}$ and $\boldsymbol{w}$ satisfy R (i.e., $R(\boldsymbol{x}, \boldsymbol{w}) = 1$). On the other hand, $\boldsymbol{x} \notin L_R$ if $R(\boldsymbol{x}, \boldsymbol{w}) = 0$ for all $\boldsymbol{w}$.

An *argument system* Π for R is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where $\mathcal{P}$ convinces $\mathcal{V}$ that $R(\boldsymbol{x}, \boldsymbol{w}) = 1$. We say Π is *interactive* if it involves interaction between $\mathcal{P}$ and $\mathcal{V}$, while Π is *non-interactive* if $\mathcal{P}$ sends a single message to $\mathcal{V}$.

A *SNARK* [7] is a non-interactive argument system that produces short proofs, as defined below.

Definition 1 (SNARK)**.** *A* succinct non-interactive argument of knowledge (SNARK) *for relation R consists of a tuple of algorithms $\Pi = (\mathcal{K}, \mathcal{P}, \mathcal{V})$:*

- $\mathcal{K}(1^\lambda, R) \to \mathsf{srs}$

  On input security parameter 1^λ and relation R, the key generation algorithm outputs the structured reference string $\mathsf{srs} = (\mathsf{pk}, \mathsf{vk})$, which includes a proving key pk and a verification key vk. We also require the key generation algorithm to output the secret trapdoor td, which is usually omitted from the notation for simplicity.

- $\mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{w}) \to \pi$

On input proving key pk*, statement* $\boldsymbol{x}$*, and witness* $\boldsymbol{w}$*, the proof generation algorithm outputs a proof* π*.*

- $\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \pi) =: b$

  On input verification key vk*, statement* $\boldsymbol{x}$*, and proof* π*, the verification algorithm outputs a bit* b*, indicating whether the proof is valid.*

A SNARK Π should be *succinct*, *complete*, *knowledge-sound*, and optionally, *zero-knowledge*.

SUCCINCTNESS. Succinctness holds if the size of any proof π satisfies

$$|\pi| = \mathrm{poly}(\lambda)\,\mathrm{polylog}(|\boldsymbol{x}| + |\boldsymbol{w}|)$$

COMPLETENESS. Completeness holds if for every pair of $(\boldsymbol{x}, \boldsymbol{w})$ such that $R(\boldsymbol{x}, \boldsymbol{w}) = 1$,

$$\Pr\left[\mathsf{srs} \leftarrow \mathcal{K}(1^\lambda, R), \pi \leftarrow \mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{w}) : \mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \pi) = 1\right] = 1$$

KNOWLEDGE SOUNDNESS. Knowledge soundness holds if for every polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time extractor Ext such that for all input randomness r,

$$\Pr\left[\begin{array}{l}(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{K}(1^\lambda, R) \\ (\boldsymbol{x}, \pi) := \mathcal{A}(\mathsf{srs}; r) \\ \boldsymbol{w} := \mathsf{Ext}(\mathsf{srs}, \mathsf{td}; r) \\ \mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \pi) = 1\end{array} : R(\boldsymbol{x}, \boldsymbol{w}) = 0\right] \leq \varepsilon(\lambda)$$

ZERO-KNOWLEDGE. Intuitively, Π is zero-knowledge (*i.e.*, Π is a zkSNARK) if, even without knowing the witness $\boldsymbol{w}$, it is still possible to simulate a proof that is indistinguishable from honestly generated ones.

Formally, statistical (or computational) zero-knowledge holds if there exists a simulator Sim such that for every unbounded (or polynomial-time) distinguisher $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{K}(1^\lambda, R) \\ (\boldsymbol{x}, \boldsymbol{w}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ \pi \leftarrow \mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{w})\end{array} : \mathcal{A}(\pi) = 1\right] \approx \Pr\left[\begin{array}{l}(\mathsf{srs}, \mathsf{td}) \leftarrow \mathcal{K}(1^\lambda, R) \\ (\boldsymbol{x}, \boldsymbol{w}) \leftarrow \mathcal{A}(\mathsf{srs}) \\ \pi \leftarrow \mathsf{Sim}(\mathsf{pk}, \mathsf{td}, \boldsymbol{x})\end{array} : \mathcal{A}(\pi) = 1\right]$$

Below we introduce two types of SNARKs that are going to be used in our construction, namely, *CP-SNARKs* and *lookup arguments*.

It is desirable if a SNARK for R can be augmented in the following way: when proving $R(\boldsymbol{x}, \boldsymbol{w}) = 1$, we can additionally demonstrate that the commitment to some portion of $\boldsymbol{w}$ is c. *CP-SNARKs* [17, 18] are proposed to achieve this goal.

Generally, CP-SNARKs consider $\boldsymbol{w} = ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega}) = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{\ell-1}, \boldsymbol{\omega})$, and $\boldsymbol{c} = (c_0, \ldots, c_{\ell-1})$, where the i-th c_i is claimed to be the commitment to the i-th vector $\boldsymbol{v}_i$. Now, the original relation we are interested in becomes $R(\boldsymbol{x}, ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega}))$, and the augmented relation that we aim to prove is $R^{\mathsf{cp}}\left((\boldsymbol{x}, \boldsymbol{c}), ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega})\right)$, which returns 1 if $\left(\boldsymbol{x}, ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega})\right)$ satisfies R and c_i is indeed the commitment to $\boldsymbol{v}_i$. Formally, $R^{\mathsf{cp}}\left((\boldsymbol{x}, \boldsymbol{c}), ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega})\right) = 1$ if and only if

$$R\left(\boldsymbol{x}, ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega})\right) = 1 \wedge \bigwedge_{i \in [0, \ell-1]} \mathsf{CM}.\mathcal{V}(\mathsf{ck}, c_i, \boldsymbol{v}_i) = 1$$

Definition 2 (CP-SNARK)**.** *For a commitment scheme* CM*, a commitment key* $\mathsf{ck} \leftarrow \mathsf{CM}.\mathcal{K}(1^\lambda)$*, and a relation* R *for statement* $\boldsymbol{x}$ *and witness* $\boldsymbol{w} = ((\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega})$*, a* commit-and-prove SNARK (CP-SNARK) *for* R *is a SNARK for relation* R^{cp} *(as defined above). A CP-SNARK consists of a tuple of algorithms* $\mathsf{CP} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$*:*

- $\mathcal{K}(1^\lambda, \mathsf{ck}, R) \rightarrow (\mathsf{pk}, \mathsf{vk})$

  On input the security parameter λ*, a commitment key* ck*, and a relation* R*, the key generation algorithm outputs a pair of proving and verification key* $\mathsf{srs} = (\mathsf{pk}, \mathsf{vk})$*.*

- $\mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{c}, (\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega}) \rightarrow \pi$

  On input the proving key pk*, the statement* $\boldsymbol{x}$*, the commitments* $\boldsymbol{c} = (c_i)_{i=0}^{\ell-1}$*, and the witness* $(\boldsymbol{v}_i)_{i=0}^{\ell-1}, \boldsymbol{\omega}$*, the proof generation algorithm outputs a proof* π*.*

- $\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{c}, \pi) =: b$

  On input the verification key vk*, the statement* $\boldsymbol{x}$*, the commitments* $\boldsymbol{c}$*, and a proof* π*, the verification algorithm outputs a bit* b*, indicating whether the proof is valid.*

If a CP-SNARK for R further satisfies zero-knowledge (*i.e.*, if it is a zkSNARK for R^{cp}), then we denote it by $\mathsf{ZKCP} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$.

We are also interested in *lookup arguments* [8, 32, 84, 80], which prove that all elements in a list of *queries* $\boldsymbol{\alpha} = (\alpha_i)_{i=0}^{\mu-1}$, are in a *lookup table* $\boldsymbol{\tau} = (\tau_j)_{j=0}^{\nu-1}$. Formally, we consider a lookup relation $R^{\mathsf{lookup}}(\boldsymbol{\tau}, \boldsymbol{\alpha})$, which returns 1 if and only if

$$\{\alpha_i\}_{i=0}^{\mu-1} \subseteq \{\tau_j\}_{j=0}^{\nu-1}$$

Definition 3 (Lookup Arguments)**.** *A* lookup argument *is a SNARK for relation* R^{lookup} *(as defined above).*

2.4 Folding Schemes

Intuitively, a non-interactive folding scheme [60] for relation R folds two instances into a single instance such that the correctness of the folded instance implies that of the original ones.

Definition 4 (NIFS)**.** *A* non-interactive folding scheme (NIFS) *consists of a tuple of algorithms* $\mathsf{NIFS} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$*:*

- $\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$

  On input security parameter 1^λ*, the setup algorithm outputs public parameters* pp*.*

- $\mathcal{K}(\mathsf{pp}, R) =: (\mathsf{pk}, \mathsf{vk})$

  On input public parameters pp *and a relation* R*, the key generation algorithm outputs a pair of proving key* pk *and verification key* vk*.*

- $\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) \to (\mathbb{U}, \mathbb{W}, \overline{T})$

  On input the proving key pk *and two instance-witness pairs* $(\mathbb{U}_1, \mathbb{W}_1)$ *and* $(\mathbb{U}_2, \mathbb{W}_2)$*, the proof generation algorithm outputs a folded instance-witness pair* $(\mathbb{U}, \mathbb{W})$ *and a folding proof* $\overline{T}$*.*

- $\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T}) =: \mathbb{U}$

  On input of the verification key vk*, two instances* $\mathbb{U}_1$ *and* $\mathbb{U}_2$*, and the folding proof* $\overline{T}$*, the verification algorithm outputs a folded instance* $\mathbb{U}$*.*

A folding scheme NIFS satisfies the following properties.
Completeness. (Perfect) completeness holds if for every PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (R, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) \leftarrow \mathcal{A}(\mathsf{pp}) \\ R(\mathbb{U}_1, \mathbb{W}_1) = 1, R(\mathbb{U}_2, \mathbb{W}_2) = 1 \\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, R) \\ (\mathbb{U}_{\mathcal{P}}, \mathbb{W}, \overline{T}) \leftarrow \mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) \\ \mathbb{U}_{\mathcal{V}} := \mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T}) : \\ \qquad\qquad \mathbb{U}_{\mathcal{P}} = \mathbb{U}_{\mathcal{V}} \wedge R(\mathbb{U}_{\mathcal{P}}, \mathbb{W}) = 1 \end{array}\right] = 1$$

Knowledge Soundness. Knowledge soundness holds if, for every polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time extractor Ext such that for all input randomness r,

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (R, \mathbb{U}_1, \mathbb{U}_2, \mathbb{W}, \overline{T}) := \mathcal{A}(\mathsf{pp}; r) \\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, R) \\ \mathbb{U} := \mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T}) \\ R(\mathbb{U}, \mathbb{W}) = 1 \\ (\mathbb{W}_1, \mathbb{W}_2) := \mathsf{Ext}(\mathsf{pp}; r) : \\ \qquad R(\mathbb{U}_1, \mathbb{W}_1) = 0 \vee R(\mathbb{U}_2, \mathbb{W}_2) = 0 \end{array}\right] \leq \varepsilon(\lambda)$$

Zero-knowledge. Statistical (or computational) zero-knowledge holds if there exists a simulator Sim such that for every unbounded (or polynomial-time) distinguisher $\mathcal{A}$,

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (R, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) \leftarrow \mathcal{A}(\mathsf{pp}) \\ R(\mathbb{U}_1, \mathbb{W}_1) = 1 \wedge R(\mathbb{U}_2, \mathbb{W}_2) = 1 \\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, R) \\ (\cdot, \cdot, \overline{T}) \leftarrow \mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) : \\ \qquad\qquad\qquad\qquad \mathcal{A}(\overline{T}) = 1 \end{array}\right]$$
$$\approx \Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda) \\ (R, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)) \leftarrow \mathcal{A}(\mathsf{pp}) \\ R(\mathbb{U}_1, \mathbb{W}_1) = 1 \wedge R(\mathbb{U}_2, \mathbb{W}_2) = 1 \\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, R) \\ \overline{T} \leftarrow \mathsf{Sim}(\mathsf{pk}, \mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2) : \\ \qquad\qquad\qquad\qquad \mathcal{A}(\overline{T}) = 1 \end{array}\right]$$

2.5 Incrementally Verifiable Computation

Incrementally verifiable computation [83] allows one to verify the repeated execution of a function $\mathcal{F}$, dubbed *step function*. Specifically, the prover can generate a proof π_i demonstrating 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 $\boldsymbol{z}_{i-1}$. This notion is formalized as below.

Definition 5 (IVC)**.** *An* incrementally verifiable computation (IVC) *scheme is composed of four algorithms* $\mathsf{IVC} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$*:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$

  On input security parameter 1^λ*, the setup algorithm* $\mathcal{G}$ *outputs the public parameters* pp*.*

- $\mathcal{K}(\mathsf{pp}, \mathcal{F}) =: (\mathsf{pk}, \mathsf{vk})$

  On input public parameters pp *and a polynomial-time computable function* $\mathcal{F}$*, the key generation algorithm* $\mathcal{K}$ *outputs a pair of proving key* pk *and verification key* vk*.*

- $\mathcal{P}(\mathsf{pk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \mathsf{aux}_i, \pi_i) \to \pi_{i+1}$

  On input the proving key pk*, an index* i*, an initial input* $\boldsymbol{z}_0$*, the claimed output* $\boldsymbol{z}_i$ *in the* i*-th iteration, the non-deterministic advice* aux_i*, and a proof* π_i *attesting to* $\boldsymbol{z}_i$*, the proof generation algorithm* $\mathcal{P}$ *outputs a new proof* π_{i+1} *that attests to* $\boldsymbol{z}_{i+1} = \mathcal{F}(\boldsymbol{z}_i; \mathsf{aux}_i)$*.*

- $\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) =: b$

  On input the verification key vk*, an index* i*, an initial input* $\boldsymbol{z}_0$*, the claimed output* $\boldsymbol{z}_i$ *in the* i*-th iteration, and a proof* π_i *attesting to* $\boldsymbol{z}_i$*, the verification algorithm* $\mathcal{V}$ *outputs a bit* b*, indicating whether the proof is valid.*

An IVC scheme IVC satisfies the following properties.
Completeness. (Perfect) completeness holds if for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathcal{G}(\lambda),\\ (\mathcal{F}, i, \boldsymbol{z}_0, \boldsymbol{z}_i, \mathsf{aux}_i, \pi_i) \leftarrow \mathcal{A}(\mathsf{pp})\\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, \mathcal{F})\\ \boldsymbol{z}_{i+1} := \mathcal{F}(\boldsymbol{z}_i; \mathsf{aux}_i)\\ \mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1\\ \pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \mathsf{aux}_i, \pi_i) :\\ \qquad \mathcal{V}(\mathsf{vk}, (i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1}), \pi_{i+1}) = 1\end{array}\right] = 1$$

Knowledge Soundness[1]. Knowledge soundness holds if, for every polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time extractor Ext such that for all input randomness r,

[1]We adopt [72]'s definition of knowledge soundness, which implies the notions in [60, 57, 11] that require Ext to extract all previous auxiliary values.

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(\lambda), \\ (\mathcal{F}, (i \geq 1, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) := \mathcal{A}(\mathsf{pp}; r) \\ (\mathsf{pk}, \mathsf{vk}) := \mathcal{K}(\mathsf{pp}, \mathcal{F}) \\ \mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1 \\ (\boldsymbol{z}_{i-1}, \mathsf{aux}_{i-1}, \pi_{i-1}) := \mathsf{Ext}(\mathsf{pp}; r) : \\ \hfill \boldsymbol{z}_i = \mathcal{F}(\boldsymbol{z}_{i-1}; \mathsf{aux}_{i-1}) \\ \hfill \wedge \mathcal{V}(\mathsf{vk}, (i-1, \boldsymbol{z}_0, \boldsymbol{z}_{i-1}), \pi_{i-1}) = 1 \end{array}\right] \geq 1 - \varepsilon(\lambda)$$

Succinctness. Succinctness holds if the size of π_i and the run time of $\mathcal{P}$ and $\mathcal{V}$ are independent of the number of iterations.

Note that, unlike the definition of succinctness in SNARKs, a succinct IVC may have proof size and verifier time that are linear in the size of $\mathcal{F}$. In addition, an IVC is not necessarily zero-knowledge.

To achieve a fully succinct and zero-knowledge IVC, one can include an additional zkSNARK that compresses the proof while hiding the witnesses [60, 11]. This concept is formalized as *decider.*

In the decider, we are interested in a relation R^{Decider} that encodes the IVC's verification algorithm. Formally, given statement $\boldsymbol{x} = (k, \boldsymbol{z}_0, \boldsymbol{z}_k)$ and witness $\boldsymbol{w} = (\pi_k)$, $R^{\mathsf{Decider}}(\boldsymbol{x}, \boldsymbol{w}) = 1$ if and only if $\mathsf{IVC}.\mathcal{V}(\mathsf{vk}_\Phi, (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k) = 1$, where vk_Φ is the IVC verification key. With this relation, we can define a decider as follows, who has the same syntax and security properties as a zkSNARK.

Definition 6 (Decider)**.** *For an IVC scheme* IVC *whose verification algorithm* $\mathsf{IVC}.\mathcal{V}$ *is expressed as a relation* R^{Decider} *(as defined above), a step function* $\mathcal{F}$*, and a pair of proving and verification key* $(\mathsf{pk}_\Phi, \mathsf{vk}_\Phi) \leftarrow \mathsf{IVC}.\mathcal{K}(\mathsf{pp}, \mathcal{F})$*, a decider* $\mathsf{Decider} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ *is a zkSNARK for* R^{Decider}.

Note that in our definition, the decider requires circuit-specific setup, which suffices for our application. That is, every structured reference string srs generated by $\mathsf{Decider}.\mathcal{K}$ is only for a specific $\mathcal{F}$ and vk_Φ.

3 Proofs of Video Authenticity

In this section, 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 as well as the security properties.

3.1 Data Types and Operations

We first consider two forms of *video* data: the raw video $\boldsymbol{V}$ and the video stream ζ.

Raw Video. A *raw video* is usually 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 two dimensional matrix of *pixels.*

A pixel consists of several *components* that carry the properties of the pixel's color or luminance. For instance, three color components R, G, and B that respectively indicate the relative proportions of red, green, and blue make up the RGB color space. It is more common to use the YCbCr color space in video processing, where a pixel is represented by a luma component Y, a blue chroma component Cb, and a red chroma component Cr, each of which is an 8-bit[2] integer.

The chroma components are usually *subsampled* in practice to reduce the data size, and we assume a $4:2:0$ subsampling ratio, where both the horizontal and vertical resolutions of Cb and Cr are halved. Hence, in a frame with M rows and N columns, there are $M \times N$ Y components, $M/2 \times N/2$ Cb components, and $M/2 \times N/2$ Cr components.

The *resolution* of a frame with M rows and N columns of pixels is defined as $N \times M$[3], where N and M are also called the *width* and the *height* of the frame. The frequency at which the frames in a video are displayed is dubbed the frame rate, which is typically measured in frames per second (FPS). High image resolution and frame rate typically appear as higher quality video.

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 $\boldsymbol{V} := (\boldsymbol{X}_i)_{i=0}^{M/16\times N/16\times L-1} \in \mathcal{B}^{M/16\times N/16\times L}$.

Video Stream. 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. As a more compact form, a video stream is then encapsulated in a *multimedia container* such as MP4, which may additionally include audio and subtitles. For simplicity, we focus solely on the visual part in this work.

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.

Now we briefly review the general workflow of macroblock-based video codecs, e.g., H.264/AVC [40], H.265/HEVC [41], and AOMedia Video 1 (AV1) [3], as illustrated in Figure 2.

The encoder aims to reduce the file size by removing *redundant* and *non-essential* information from $\boldsymbol{V}$ while maintaining as much visual quality as possible. To this end, the encoder employs four stages for every macroblock $\boldsymbol{X}$ in $\boldsymbol{V}$: *prediction*, *transform*, *quantization*, and *entropy coding*, where only quantization may introduce loss of information, while the other stages are lossless.

1. During prediction Pred, the encoder generates a *prediction macroblock* $\boldsymbol{P}$ for $\boldsymbol{X}$, so that the difference between the original macroblock and the prediction macroblock is minimized. There are two types of prediction: *intra-frame prediction* that removes spatial redundancy within a frame (e.g., background areas with uniform colors or patterns), and *inter-frame prediction* that avoids temporal redundancy among multiple frames (e.g., stationary areas with no motion or moving objects with simple patterns) by

[2]It is possible to have a higher bit depth (e.g., 10-bit) in video codecs, but we only discuss 8-bit color components for clarity.

[3]Note that while the frame resolution $N \times M$ is column-first, the frame matrix is still written in row-major order.

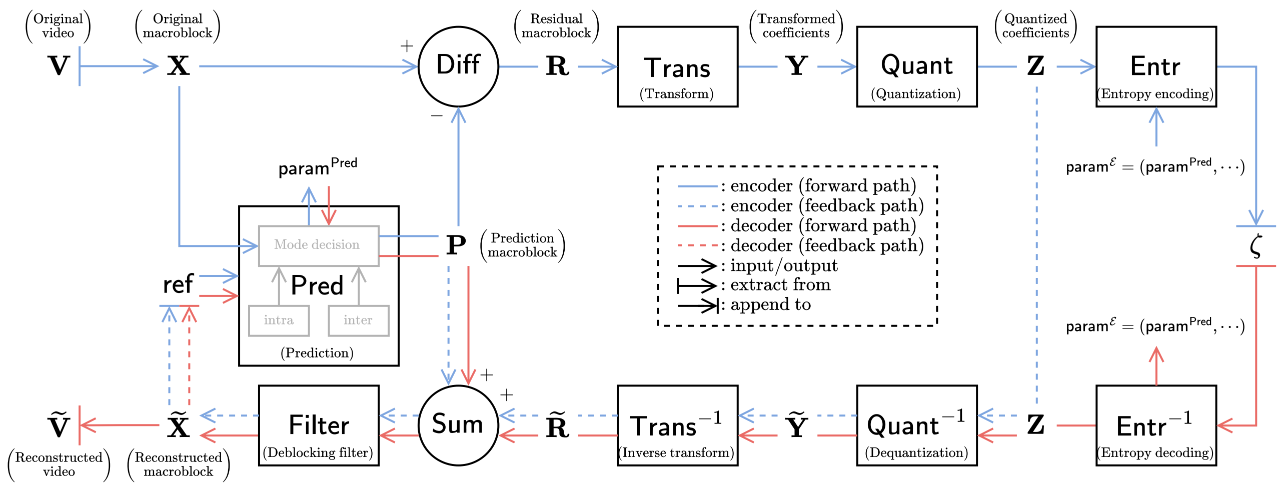


Figure 2: Block diagram of a macroblock-based video codec. Blue lines: data flow during encoding. Red lines: data flow during decoding. Solid lines: forward paths. Dashed lines: feedback paths for updating the reference information. Arrows with a vertical bar at the start: the data is extracted from the source. Arrows with a vertical bar at the end: the data is appended to the destination.

leveraging motion estimation and motion compensation. Both prediction modes rely on some reference information ref, which we will discuss later.

With all possible prediction results, we decide the final prediction mode by selecting the best result $\boldsymbol{P}$ whose difference between $\boldsymbol{X}$ is minimal. The final difference $\boldsymbol{X} - \boldsymbol{P}$ is output as the *residue macroblock* $\boldsymbol{R}$, and the prediction parameters $\mathsf{param}^{\mathsf{Pred}} = (\mathsf{mode}, \cdots)$ are also returned, where mode is the selected prediction mode (“inter” or “intra”).

2. During transform Trans, the encoder further reduces the spatial redundancy by transforming the pixel data in the residue macroblock $\boldsymbol{R}$ to the frequency domain. In this way, we can obtain the low-frequency components representing essential features and the high-frequency components containing non-essential details.

   This process usually involves Discrete Cosine Transform (DCT) and Hadamard Transform, after which the *transformed coefficients* $\boldsymbol{Y}$ are forwarded to the next stage.

3. During quantization Quant, the precision of transformed coefficients $\boldsymbol{Y}$ is reduced, in order to discard non-essential information (e.g., perceptually hard-to-notice details) in the coefficients.

   Quantization is done by scaling and rounding the coefficients, obtaining the *quantized coefficients* $\boldsymbol{Z}$, where rounding is the main reason of information loss.

4. Finally, entropy coding Entr minimizes statistical redundancy in quantized coefficients $\boldsymbol{Z}$ by assigning shorter codes to more frequent elements, whereas less frequent data is mapped to longer codes. The encoding parameters $\mathsf{param}^{\mathcal{E}}$ are also compressed by Entr, which contains parameters used in the encoding process such as the prediction parameters $\mathsf{param}^{\mathsf{Pred}}$.

Examples of entropy coding include Huffman coding and arithmetic coding. The output of entropy coding is appended to the bitstream ζ.

Note that the reference information ref used for predicting subsequent marcoblocks needs to be computed by *reconstructing* from already encoded data. This is generally done by reversing the encoding algorithm. Given the quantized coefficients $\boldsymbol{Z}$ encoded from $\boldsymbol{X}$ and its prediction $\boldsymbol{P}$, $\boldsymbol{Z}$ is first *dequantized* via Quant^{-1}, which returns the dequantized coefficients $\widetilde{\boldsymbol{Y}}$. Due to the loss of information during quantization, they are close but may not be equal to the original transformed coefficients, *i.e.*, $\widetilde{\boldsymbol{Y}} \approx \boldsymbol{Y}$. $\widetilde{\boldsymbol{Y}}$ is then *inverse transformed* via Trans^{-1} to obtain the residual macroblock $\widetilde{\boldsymbol{R}} \approx \boldsymbol{R}$. Next, we compute the sum of $\widetilde{\boldsymbol{R}}$ and the prediction macroblock $\boldsymbol{P}$, which is fed to an optional *deblocking filter* to get the reconstructed macroblock $\widetilde{\boldsymbol{X}} \approx \boldsymbol{X}$.

Reconstruction is also the core subroutine of the decoding process. Before reconstruction, the decoder extracts a subsequence from the bitstream ζ and applies entropy decoding on the subsequence to get the quantized coefficients $\boldsymbol{Z}$. The decoder then reconstructs the macroblock $\widetilde{X}$ in the same way as encoder, where the prediction macroblock $\boldsymbol{P}$ used for reconstruction is generated by performing the prediction operation Pred on input the previously reconstructed reference information ref. Since the prediction mode mode is encoded in the bitstream ζ, it is unnecesary to have $\boldsymbol{X}$ when choosing the prediction mode in Pred.

Formally, we define a block-wise encoding operation $\mathcal{E} : \mathcal{B} \times \{0,1\}^* \times \{0,1\}^* \to \mathcal{B} \times \{0,1\}^*$, which takes a macroblock $\boldsymbol{X}$, some reference information $\mathsf{ref} \in \{0,1\}^*$, and some encoding parameters $\mathsf{param}^{\mathcal{E}}$ as input, encodes $\boldsymbol{X}$ under $\mathsf{param}^{\mathcal{E}}$ with the help of ref, and outputs the reconstructed macroblock $\boldsymbol{X}'$ and the encoded bitstream $y \in \{0,1\}^*$. The encoding parameters $\mathsf{param}^{\mathcal{E}}$ control the quality and performance of the encoding process. In addition to prediction parameters $\mathsf{param}^{\mathsf{Pred}}$, we also include other configurations in $\mathsf{param}^{\mathcal{E}}$. For instance, in H.264, $\mathsf{param}^{\mathcal{E}}$ contains the quantization parameter qp, which determines the precision of quantized coefficients.

Conversely, the block-wise decoding operation $\mathcal{D} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \mathcal{B}$ takes the encoded bitstream y, the reference information ref, and the encoding parameters $\mathsf{param}^{\mathcal{E}}$ as input, decodes y under $\mathsf{param}^{\mathcal{E}}$ with the help of ref, and outputs the reconstructed macroblock $\boldsymbol{X}'$.

Abusing the notation slightly, we allow applying $\mathcal{E}$ to the entire video $\boldsymbol{V}$ to obtain the encoded video stream $\zeta := \mathcal{E}(\boldsymbol{V}, \mathsf{param}^{\mathcal{E}})$, and $\mathcal{D}$ to the encoded video stream ζ to obtain the decoded video $\boldsymbol{V}' := \mathcal{D}(\zeta, \mathsf{param}^{\mathcal{E}})$. 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 author name, 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 $\boldsymbol{X}'$. param^{Δ} contains configurations specific to the editing operation, such as the brightness level, the position of an overlay mask, etc.

While one may also edit the metadata meta of a video in practice, we assume that meta is immutable in our definition. This assumption does not invalidate editing operations such as cropping and cutting, since the resolution and frame rate are regarded as part of the encoded video stream ζ.

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., a 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., a camcorder) records a video (the raw footage), 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., a 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., a website visitor) checks if the proof is valid w.r.t. the video by executing the verification algorithm $\mathcal{V}$.

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

Definition 7 (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 the secret trapdoor td, which 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 $\mathcal{E}$, $R^{\mathsf{VA}}(\boldsymbol{x}, \boldsymbol{w}) = 1$ if and only if

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

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,

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

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,

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

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 $\mathcal{A}$,

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

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

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 [42], 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. In this section, we discuss both paradigms, and then we provide a concrete instantiation Loua by applying them to Nova [60]-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* $\widetilde{\mathcal{F}}$ 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 [60, 16] are able to convert it to an IVC scheme IVC. Both folding-to-IVC compilers are conceptually similar: to build an IVC for a step circuit $\mathcal{F}$, a "wrapper" (or formally, an *augmented step circuit*) $\widetilde{\mathcal{F}}$ is first defined, which 1) performs the current execution of $\mathcal{F}$, and 2) verifies the previous folding proof. The circuit $\widetilde{\mathcal{F}}$ is then encoded under a constraint system as a relation $\widetilde{\mathsf{CS}}$ between witness $\boldsymbol{w}$ and statement (public I/O) $\boldsymbol{x}$. Without loss of generality, we adopt the notations in [60] in the following, and we assume $\boldsymbol{w}$ is contained in the folding witness, and $\boldsymbol{x}$ is in the folding instance. Finally, the folding scheme NIFS is used to accumulate the folding witness and the folding instance in each step of the IVC scheme IVC.

Formally, the *incoming* instance-witness pair that is associated with a single execution of $\widetilde{\mathcal{F}}$ is denoted as $(\mathbbm{u}, \mathbbm{w})$, and the *running* instance-witness pair that is associated with all previous executions of $\widetilde{\mathcal{F}}$ is denoted as $(\mathbb{U}, \mathbb{W})$. For instance, in the i-th step, the incoming instance-witness pair $(\mathbbm{u}_i, \mathbbm{w}_i)$ represents the execution of $\widetilde{\mathcal{F}}$ in the $(i-1)$-th step, while the running instance-witness pair $(\mathbb{U}_i, \mathbb{W}_i)$ is the accumulation of all previous $(\mathbbm{u}_j, \mathbbm{w}_j)$ for $j \in [0, i-2]$ and thus represents all previous $i-1$ invocations (from 0-th step to $(i-2)$-th step) of $\widetilde{\mathcal{F}}$.

In each step of recursive computation, the incoming instance-witness pair is absorbed into the running instance-witness pair. More specifically, given $(\mathbbm{u}_i, \mathbbm{w}_i)$ and $(\mathbb{U}_i, \mathbb{W}_i)$, the prover $\mathsf{IVC}.\mathcal{P}$ folds both pairs, producing a new running instance-witness pair $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1})$ that represents all previous i invocations (from 0-th step to $(i-1)$-th step) of $\widetilde{\mathcal{F}}$. Next, the prover performs the i-th execution of $\widetilde{\mathcal{F}}$, whose corresponding instance-witness pair $(\mathbbm{u}_{i+1}, \mathbbm{w}_{i+1})$ is stored for the $(i+1)$-th step of $\mathsf{IVC}.\mathcal{P}$.

Here, the i-th execution of $\widetilde{\mathcal{F}}$ 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) $\mathsf{NIFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_i, \mathbb{u}_i, \overline{T}) = \mathbb{U}_{i+1}$, 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 $\widetilde{\mathcal{F}}$ in the $(i-1)$-th step. Additionally, $\widetilde{\mathcal{F}}$ 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 $\widetilde{\mathcal{F}}$ in the i-th step.

Support lookup in folding-based IVC. Now we are ready to discuss how to equip any IVC based on NIFS 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 the lookup table $\boldsymbol{\tau}$. Further, 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 $\mathsf{NIFS}^{\mathsf{LU}}$, 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 $\mathsf{NIFS}^{\mathsf{LU}}$ 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, in folding schemes where $\mathbb{U}$ already contains the commitment to $\boldsymbol{w}$ (e.g., [60, 59, 57, 11, 27]), our approach can be viewed as the separation of $\boldsymbol{q}$ from $\boldsymbol{w}$.

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

```
Fn $\mathsf{NIFS}^{\mathsf{LU}}.\mathcal{G}(1^\lambda)$:
    return $\mathsf{pp} \leftarrow \mathsf{NIFS}.\mathcal{G}(1^\lambda)$
Fn $\mathsf{NIFS}^{\mathsf{LU}}.\mathcal{K}(\mathsf{pp}, R)$:
    return $(\mathsf{pk}, \mathsf{vk}) := \mathsf{NIFS}.\mathcal{K}(\mathsf{pp}, R)$
Fn $\mathsf{NIFS}^{\mathsf{LU}}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2))$:
    $(\mathbb{U}, \mathbb{W}, \overline{T}) \leftarrow \mathsf{NIFS}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2))$
    $r := \rho(\mathsf{tr})$                              ▷ 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 $\mathsf{NIFS}^{\mathsf{LU}}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T})$:
    $\mathbb{U} := \mathsf{NIFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T})$
    $r := \rho(\mathsf{tr})$                              ▷ 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 $\mathsf{IVC}^{\mathsf{LU}}$ from $\mathsf{NIFS}^{\mathsf{LU}}$. Recall that IVC for $\mathcal{F}$ internally utilizes NIFS for an augmented step circuit $\widetilde{\mathcal{F}}$. This is also the case in our construction. Inspired by `gnark` [9], which incorporates LogUp into Groth16 [39] and Plonk [33], our augmented step circuit $\widetilde{\mathcal{F}}$ (Circuit 2) additionally checks the set inclusion identity of LogUp [42, Lemma 5] in-circuit.

Suppose $\mathcal{F}$, during its execution, 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 [31], we can eliminate the interaction and compute $c \coloneqq \rho(\overline{Q})$ instead. Consequently, the $\widetilde{\mathcal{F}}$ 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 $\widetilde{\mathcal{F}}$.

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, among the inputs to $\widetilde{\mathcal{F}}$, we only have $\mathbb{u}_i$ that represents the execution of $\widetilde{\mathcal{F}}$ in the $(i-1)$-th step, while $\mathbb{u}_{i+1}$ will be calculated *after* the current execution of $\widetilde{\mathcal{F}}$ and is unknown to $\widetilde{\mathcal{F}}$ *during* its execution.

To address this circular dependency, 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 $\widetilde{\mathcal{F}}$ 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 $\mathbb{u}_{i+1}.\boldsymbol{x}$, $\widetilde{\mathcal{F}}$ 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, $\widetilde{\mathcal{F}}$ 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: $\widetilde{\mathcal{F}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \overline{T}, \mathsf{aux}_i) \to (h, c)$

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

With $\widetilde{\mathcal{F}}$ as the augmented step circuit, we construct $\mathsf{IVC}^{\mathsf{LU}}$ (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 $\widetilde{\mathcal{F}}$ 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 $\widetilde{\mathcal{F}}$ 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 $\pi_i = ((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$ is straightforward: we simply check the digest h and challenge c in $\mathbb{u}_i.\boldsymbol{x}$ and ensure that $\mathbb{w}_i, \mathbb{W}_i$ satisfy $\mathbb{u}_i, \mathbb{U}_i$, respectively.

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

```
Fn $\mathsf{IVC}^{\mathsf{LU}}.\mathcal{G}(1^\lambda)$:
    return $\mathsf{pp} \leftarrow \mathsf{NIFS}^{\mathsf{LU}}.\mathcal{G}(1^\lambda)$
Fn $\mathsf{IVC}^{\mathsf{LU}}.\mathcal{K}(\mathsf{pp}, \mathcal{F})$:
    Wrap $\mathcal{F}$ and build $\widetilde{\mathcal{F}}$
    Encode $\widetilde{\mathcal{F}}$ as $\widetilde{\mathsf{CS}}$ in the underlying constraint system
    return $(\mathsf{pk}, \mathsf{vk}) := \mathsf{NIFS}^{\mathsf{LU}}.\mathcal{K}(\mathsf{pp}, \widetilde{\mathsf{CS}})$
Fn $\mathsf{IVC}^{\mathsf{LU}}.\mathcal{P}(\mathsf{pk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \mathsf{aux}_i, \pi_i)$:
    Parse $((\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i)) := \pi_i$
    $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}, \overline{T}) := (i = 0)\ ?\ (\mathbb{U}_\perp, \mathbb{W}_\perp, \perp) : \mathsf{NIFS}^{\mathsf{LU}}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_i, \mathbb{W}_i), (\mathbb{u}_i, \mathbb{w}_i))$
    $(h, c) \leftarrow \widetilde{\mathcal{F}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \overline{T}, \mathsf{aux}_i)$
    Construct $\mathbb{u}_{i+1}, \mathbb{w}_{i+1}$ from the execution of $\widetilde{\mathcal{F}}$
    return $\pi_{i+1} := ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbb{u}_{i+1}, \mathbb{w}_{i+1}))$
Fn $\mathsf{IVC}^{\mathsf{LU}}.\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i)$:
    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
```

By applying this paradigm to Nova and relaxed R1CS [60], we obtain a folding scheme and an IVC scheme, respectively named LouaFS and LouaIVC. We additionally utilize CycleFold [56] to improve circuit efficiency. The full constructions of LouaFS and LouaIVC, together with discussion on their security, can be found in Appendices B.1 and B.2.

Comparison with other IVC that supports lookup. Several folding-based IVC schemes [57, 11, 12, 58] also support lookup arguments. In HyperNova [57], the authors build nlookup upon the sum-check protocol. Similar to our paradigm, Protostar [11] and its subsequent work [12] utilizes LogUp as well. Very recently, NeutronNova [58] integrates Lasso [80] with folding.

For the use cases where the lookup table size ν is large, [57, 11, 12] 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 $\mathsf{Protostar}$'s solution can be combined with folding schemes that are not based on special-sound protocols [11].

4.2 Paradigm 2: Commit-and-Prove Decider

We introduce a decider $\mathsf{Decider}$ 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. Before diving into the details, we first review two different methods for building $\mathsf{Decider}$ upon zkSNARKs [60, 75].

In Nova [60], the authors construct a dedicated Polynomial IOP [13] 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 [14] as the IPA, and a two-tiered PCS (e.g., Dory-PC [61]) with Dory-IPA [61]. For an augmented step circuit $\widetilde{\mathcal{F}}$ 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` [75] instead expresses R^{IVC} as a circuit and prove its satisfiability with Groth16 [39], yielding constant proof size and verifier time. Nevertheless, `sonobe`'s decider only supports compressing proofs that use KZG commitment [49], 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 $\mathsf{NIFS}^{\mathsf{LU}}$) 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 polynomial interpolation during the conversion).

Our approach leverages CP-SNARKs [17], 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 [73] as CM by choosing $\mathsf{LegoGro16}$ [17] 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 [60, 75] 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 $(\mathtt{u}_k, \mathtt{w}_k)$ 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 $\mathsf{Decider}$ in Algorithm 4. As discussed above, $\mathsf{Decider}.\mathcal{P}$ first runs $\mathsf{NIFS}.\mathcal{P}$ to fold $(\mathtt{u}_k, \mathtt{w}_k)$ into $(\mathbb{U}_k, \mathbb{W}_k)$, and then generates a proof ϖ with $\mathsf{ZKCP}.\mathcal{P}$, attesting that $\mathcal{F}^{\mathsf{Decider}}$ is satisfiable and that the commitments in $\mathbb{U}_{k+1}$ are valid. The verifier $\mathsf{Decider}.\mathcal{V}$ folds $\mathtt{u}_k$ into $\mathbb{U}_k$ as well, checks the public inputs in $\mathbb{u}_k.\boldsymbol{x}$, and then verifies the proof ϖ and the commitments in $\mathbb{U}_{k+1}$ using ZKCP.$\mathcal{V}$.

It is worth noting that, due to Groth16, 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 IVC.$\mathcal{P}$, and it thus can be relatively cheap in practice.

Algorithm 4: Decider

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

Loua's decider LouaDecider can be constructed with LouaFS and LegoGro16, and we provide its details in Appendix B.3, together with the construction of the associated circuit $\mathcal{F}^{\mathsf{Decider}}$ and the security proof.

5 The Eva Protocol

In this section, we introduce the construction of Eva, our proof of video authenticity based on IVC.

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 sub-procedures on each macroblock of the video.

Thus, we first construct the gadgets for encoding and editing a single macroblock. In Section 5.1, we elaborate on $\mathcal{F}^{\mathcal{E}}$, the gadget for video encoding, as well as its building blocks. In Section 5.2, we present instantiations of $\mathcal{F}^{\Delta}$ for several video editing operations.

Then we provide the construction of our IVC step circuit $\mathcal{F}^{\mathsf{Eva}}$ in Section 5.3, where we discuss how to integrate the two key components $\mathcal{F}^{\mathcal{E}}$ and $\mathcal{F}^{\Delta}$ into $\mathcal{F}^{\mathsf{Eva}}$, which, at the same time, checks the validity of signature σ.

Finally, in Section 5.4, we build upon $\mathcal{F}^{\mathsf{Eva}}$ the full construction of Eva by naturally extending the step circuit to handle the entire video $\boldsymbol{V}$, with the help of our folding-based IVC Loua. We end this section with the discussion on the security of Eva.

5.1 Gadgets for Video Encoding

First, we construct $\mathcal{F}^{\mathcal{E}}$, a gadget for encoding a single macroblock in a video. Specifically, we focus on supporting H.264/AVC [40], but our methodology can be extended to other block-based video codecs such as H.265/HEVC, AV1, etc.

Naively, one may translate the entire encoding algorithm $\mathcal{E}$ to the $\mathcal{F}^{\mathcal{E}}$ gadget. However, due to the reasons below, this would require a prohibitive number of constraints, which is infeasible in practice.

- The encoding process involves complex operations, such as motion estimation, entropy coding, etc.
- The encoding of a macroblock may depend on other macroblocks in the current frame (when the prediction mode is $\mathsf{mode} =$ "intra") or even in the neighboring frames (when $\mathsf{mode} =$ "inter").

To address these challenges, we make extensive use of verifier's knowledge. Note that 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, as depicted in Figure 3, while it becomes unnecessary to prove prediction and entropy coding.

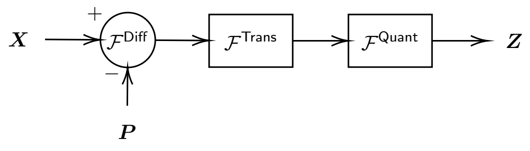

$\boldsymbol{X}$
+
$\mathcal{F}^{\mathsf{Diff}}$
−
$\boldsymbol{P}$
$\mathcal{F}^{\mathsf{Trans}}$
$\mathcal{F}^{\mathsf{Quant}}$
$\boldsymbol{Z}$

Figure 3: Overview of in-circuit operations for video encoding

As summarized in Gadget 5, the encoding gadget $\mathcal{F}^{\mathcal{E}}$ takes as input the current macroblock $\boldsymbol{X}$, the current prediction macroblock $\boldsymbol{P}$, and additionally the encoding parameters $\mathsf{param}^{\mathcal{E}}$, and returns the quantized coefficients $\boldsymbol{Z}$ by running the gadgets $\mathcal{F}^{\mathsf{Diff}}$, $\mathcal{F}^{\mathsf{Trans}}$, and $\mathcal{F}^{\mathsf{Quant}}$. Here, $\mathcal{F}^{\mathsf{Diff}}$ 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 the following sections.

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

```
1 $\boldsymbol{R} := \mathcal{F}^{\mathsf{Diff}}(\boldsymbol{X}, \boldsymbol{P})$          ▷ Compute residual macroblock $\boldsymbol{R}$
2 $\boldsymbol{Y} := \mathcal{F}^{\mathsf{Trans}}(\boldsymbol{R})$          ▷ Compute transformed coefficients $\boldsymbol{Y}$
3 $\boldsymbol{Z} \leftarrow \mathcal{F}^{\mathsf{Quant}}(\boldsymbol{Y}, \mathsf{param}^{\mathcal{E}})$   ▷ Compute quantized coefficients $\boldsymbol{Z}$
4 return $\boldsymbol{Z}$
```

5.1.1 Transform

The transform operation in H.264 is based on 4×4 DCT (Discrete Cosine Transform), but with slight difference for efficiency reasons: it only involves integer operations, while the fractional part of the DCT coefficients is handled in the quantization step. More specifically, with the core transform matrix $\boldsymbol{C}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$, the transform on a 4×4 block $\boldsymbol{\Lambda}$ is computed as $\boldsymbol{\Gamma} := \boldsymbol{C}_4 \boldsymbol{\Lambda} \boldsymbol{C}_4^\intercal$. Hence, the construction of the corresponding gadget $\mathcal{F}^{\mathsf{Trans}}$ is straightforward: for a residual macroblock $\boldsymbol{R}$ with color components $\boldsymbol{R}^{\mathsf{Y}}, \boldsymbol{R}^{\mathsf{Cb}}, \boldsymbol{R}^{\mathsf{Cr}}$, we divide them into blocks of 4×4 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 the Hadamard matrices $\boldsymbol{H}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$ and $\boldsymbol{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, respectively. Finally, the transformed DC coefficients $\boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}}$ as well as the AC coefficients $(\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}$, $(\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}$ are returned. The entire in-circuit transform process is depicted in Gadget 6.

Gadget 6: $\mathcal{F}^{\mathsf{Trans}}(\boldsymbol{R})$

```
1  $\boldsymbol{C}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}, \boldsymbol{H}_4 := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}, \boldsymbol{H}_2 := \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
2  for $k \in [0, 16)$ do
3      $i := \lfloor k/4 \rfloor, j := k \bmod 4$
4      $\boldsymbol{A}_k^{\mathsf{Y}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Y}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^\intercal$
5      $b_{i,j}^{\mathsf{Y}} := a_{k,0,0}^{\mathsf{Y}}$
6  for $k \in [0, 4)$ do
7      $i := \lfloor k/2 \rfloor, j := k \bmod 2$
8      $\boldsymbol{A}_k^{\mathsf{Cb}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Cb}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^\intercal$
9      $\boldsymbol{A}_k^{\mathsf{Cr}} := \boldsymbol{C}_4 \boldsymbol{R}^{\mathsf{Cr}}[4i, 4i+4; 4j, 4j+4] \boldsymbol{C}_4^\intercal$
10     $b_{i,j}^{\mathsf{Cb}} := a_{k,0,0}^{\mathsf{Cb}}$
11     $b_{i,j}^{\mathsf{Cr}} := a_{k,0,0}^{\mathsf{Cr}}$
12 $\boldsymbol{D}^{\mathsf{Y}} := \boldsymbol{H}_4 \boldsymbol{B}^{\mathsf{Y}} \boldsymbol{H}_4^\intercal$
13 $\boldsymbol{D}^{\mathsf{Cb}} := \boldsymbol{H}_2 \boldsymbol{B}^{\mathsf{Cb}} \boldsymbol{H}_2^\intercal$
14 $\boldsymbol{D}^{\mathsf{Cr}} := \boldsymbol{H}_2 \boldsymbol{B}^{\mathsf{Cr}} \boldsymbol{H}_2^\intercal$
15 $\boldsymbol{Y} := ((\boldsymbol{A}_i^{\mathsf{Y}})_{i=0}^{15}, (\boldsymbol{A}_i^{\mathsf{Cb}})_{i=0}^{4}, (\boldsymbol{A}_i^{\mathsf{Cr}})_{i=0}^{4}, \boldsymbol{D}^{\mathsf{Y}}, \boldsymbol{D}^{\mathsf{Cb}}, \boldsymbol{D}^{\mathsf{Cr}})$
16 return $\boldsymbol{Y}$
```

5.1.2 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. A large qp leads to a higher compression ratio but also more distortion, while a small qp results in larger file sizes but better quality.

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. The shift δ is $15 + \lfloor \mathsf{qp}/6 \rfloor$ for AC coefficients and $16 + \lfloor \mathsf{qp}/6 \rfloor$ for DC coefficients.

In H.264, floating-point operations such as rounding are further replaced with approximate integer operations, as the latter are more efficient in hardware. This is also beneficial for reducing the circuit size: even with the state-of-the-art techniques [28], the in-circuit floating-point operations are still expensive, e.g., a single FP32 division would cost ~ 76 constraints. Now, the absolute value and the sign of the quantized coefficient u is computed as $\begin{cases} \mathrm{abs}(u) := (\mathrm{abs}(v) \times \psi + \phi) \gg \delta \\ \mathrm{sign}(u) := \mathrm{sign}(v) \end{cases}$, where v's absolute value is scaled by the multiplication factor ψ, added with an offset ϕ, and then right shifted by δ bits. The offset ϕ equals f for AC coefficients, and is $2f$ for DC coefficients, where $f := 2^{4+\lfloor \mathsf{qp}/6 \rfloor} \times \begin{cases} 682, & \mathsf{mode} = \text{“intra”} \\ 342, & \mathsf{mode} = \text{“inter”} \end{cases}$. The values 682 and 342 are taken from the H.264 `JM` reference software [68], which are the approximate values of $2^{11}/3$ and $2^{11}/6$ respectively.

When dealing with scaling and rounding in-circuit, we leverage the efficient gadgets $\mathcal{F}^{\mathsf{SignAbs}}$ and $\mathcal{F}^{\gg}$ in [28] for computing absolute values and shifting operations. Both gadgets make queries to a lookup table, which can be efficiently checked by our IVC with lookup arguments integrated. The constructions of these gadgets are given in Appendix C. 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 7.

Gadget 7: $\mathcal{F}^{\mathsf{ScaleRound}}(v, \psi, \phi, \delta)$

```
1 $(s, u) \leftarrow \mathcal{F}^{\mathsf{SignAbs}}(v)$
2 $t := \mathcal{F}^{\gg}(u \times \psi + \phi, \delta)$
3 $v := s \mathbin{?} t : -t$
```

Now we are finally ready to present the quantization gadget $\mathcal{F}^{\mathsf{Quant}}$. As per `JM`, the matrix of multiplication factors is defined as $\boldsymbol{\Psi} := \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix}$. 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 $\boldsymbol{\Psi}$, where $p_{i,j} = \begin{cases} 0, & (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\} \\ 1, & (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\} \\ 2, & \text{otherwise} \end{cases}$. We use a matrix $\boldsymbol{P}$ to represent the mapping from (i, j) to $p_{i,j}$. 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.

The entire quantization process is summarized in Gadget 8, with qp and mode included in $\mathsf{param}^{\mathcal{E}}$.

5.2 Gadgets for Video Editing

We showcase various gadgets for video editing, including color manipulations (e.g., conversion to grayscale, brightness adjustment, color inversion), spatial operations (e.g., masking, cropping), and temporal operations (e.g., cutting).

Gadget 8: $\mathcal{F}^{\mathsf{Quant}}(\boldsymbol{Y}, \mathsf{param}^{\mathcal{E}})$

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

Additionally, we explain how to perform complex editing operations that involve multiple macroblocks in-circuit.

5.2.1 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, converting pixels in RGB to grayscale requires computing the luminance from the RGB components, which is given by $0.299 \times \mathsf{R} + 0.587 \times \mathsf{G} + 0.114 \times \mathsf{B}$. In YCbCr, the luma component already represents the luminance, and thus we can simply keep the luma component unchanged while setting the chroma components to 128.

We depict the grayscale conversion gadget for a macroblock $\boldsymbol{X}$ in Gadget 9.

Gadget 9: $\mathcal{F}^{\Delta_{\mathsf{gray}}}(\boldsymbol{X})$

```
1  Parse $(\boldsymbol{X}^{\mathsf{Y}}, \cdot) := \boldsymbol{X}$
2  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 10. We support 65536 levels of brightness adjustment, with $\mathsf{param}^{\mathsf{bright}} \in \{\frac{0}{256}, \frac{1}{256}, \ldots, \frac{65535}{256}\}$. Given a luma component x^{Y} and a parameter $\mathsf{param}^{\mathsf{bright}} = \frac{\beta}{256}$, we handle the in-circuit scaling operation by first computing $\beta \times x^{\mathsf{Y}}$ and then right shifting the product by 8 bits. Next, in order to clamp the product to $[0, 255]$, we again shift the result to the right by 8 bits. If the remaining bits are all 0, then the original result is returned as it is smaller than 256. Otherwise, we return 255.

Gadget 10: $\mathcal{F}^{\Delta_{\mathsf{bright}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{bright}} = \frac{\beta}{256})$

```
1 Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) \coloneqq \boldsymbol{X}$
2 for $i \in [0, 16), j \in [0, 16)$ do
3     $u \coloneqq \mathcal{F}^{\gg}(x^{\mathsf{Y}}_{i,j} \times \beta, 8)$
4     $v \coloneqq \mathcal{F}^{\gg}(u, 8)$
5     $x^{\mathsf{Y}}_{i,j} \coloneqq (v = 0) \mathbin{?} u : 255$
6 return $\boldsymbol{X}' \coloneqq (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

Gadget 11 illustrates the gadget for color inversion, where we subtract all components in each pixel value from 255.

Gadget 11: $\mathcal{F}^{\Delta_{\mathsf{inv}}}(\boldsymbol{X})$

```
1 Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) \coloneqq \boldsymbol{X}$
2 for $i \in [0, 16), j \in [0, 16)$ do
3     $x^{\mathsf{Y}}_{i,j} \coloneqq 255 - x^{\mathsf{Y}}_{i,j}$
4 for $i \in [0, 8), j \in [0, 8)$ do
5     $x^{\mathsf{Cb}}_{i,j} \coloneqq 255 - x^{\mathsf{Cb}}_{i,j}$
6     $x^{\mathsf{Cr}}_{i,j} \coloneqq 255 - x^{\mathsf{Cr}}_{i,j}$
7 return $\boldsymbol{X}' \coloneqq (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

5.2.2 Spatial and Temporal Operations

Now we present the gadgets for spatial and temporal operations.

To mask a macroblock $\boldsymbol{X}$ with a layer $\boldsymbol{L}$, we additionally require a binary matrix $\boldsymbol{B}$, where each bit $b_{i,j}$ indicates whether we should replace the pixel in $\boldsymbol{X}$ with the corresponding pixel in $\boldsymbol{L}$. More specifically, if $b_{i,j}$ is true, then $x_{i,j}$ is updated to $l_{i,j}$, while $x_{i,j}$ remains unchanged otherwise. 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 12. 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 12: $\mathcal{F}^{\Delta_{\mathsf{mask}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{mask}} = (\boldsymbol{B}, \boldsymbol{L}))$

```
1 Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) \coloneqq \boldsymbol{X}$
2 Parse $(\boldsymbol{B}^{\mathsf{Y}}, \boldsymbol{B}^{\mathsf{Cb}}, \boldsymbol{B}^{\mathsf{Cr}}) \coloneqq \boldsymbol{B}$
3 Parse $(\boldsymbol{L}^{\mathsf{Y}}, \boldsymbol{L}^{\mathsf{Cb}}, \boldsymbol{L}^{\mathsf{Cr}}) \coloneqq \boldsymbol{L}$
4 for $i \in [0, 16), j \in [0, 16)$ do
5     $x^{\mathsf{Y}}_{i,j} \coloneqq b^{\mathsf{Y}}_{i,j} \mathbin{?} l^{\mathsf{Y}}_{i,j} : x^{\mathsf{Y}}_{i,j}$
6 for $i \in [0, 8), j \in [0, 8)$ do
7     $x^{\mathsf{Cb}}_{i,j} \coloneqq b^{\mathsf{Cb}}_{i,j} \mathbin{?} l^{\mathsf{Cb}}_{i,j} : x^{\mathsf{Cb}}_{i,j}$
8     $x^{\mathsf{Cr}}_{i,j} \coloneqq b^{\mathsf{Cr}}_{i,j} \mathbin{?} l^{\mathsf{Cr}}_{i,j} : x^{\mathsf{Cr}}_{i,j}$
9 return $\boldsymbol{X}' \coloneqq (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

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 13, 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 5.3.

Gadget 13: $\mathcal{F}^{\Delta_{\mathsf{remove}}}(\boldsymbol{X}, \mathsf{param}^{\mathsf{remove}} = b)$

```
1 Parse $(\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}}) \coloneqq \boldsymbol{X}$
2 return $\boldsymbol{X}' \coloneqq b \,?\, (\perp, \perp, \perp) : (\boldsymbol{X}^{\mathsf{Y}}, \boldsymbol{X}^{\mathsf{Cb}}, \boldsymbol{X}^{\mathsf{Cr}})$
```

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.

5.2.3 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 can leverage vector commitment schemes [20] such as 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.

5.3 Building the Step Circuit

With the gadgets for video encoding and editing in place, we are now ready to construct the step circuit $\mathcal{F}^{\mathsf{Eva}}$. We discuss how $\mathcal{F}^{\mathsf{Eva}}$ achieves the proof of correct editing and encoding and the proof of valid signature separately.
Proof of Editing and Encoding. First, we integrate $\mathcal{F}^{\Delta}$ and $\mathcal{F}^{\mathcal{E}}$ into $\mathcal{F}^{\mathsf{Eva}}$ to guarantee the honesty of editing and encoding. 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 then 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 $\boldsymbol{Z}_j$. As we will see in Section 6 and Section 7, with a reasonably large b, we can amortize the constraints for folding verification in the augmented circuit, 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 $\mathsf{IVC}^{\mathsf{LU}}$ (and hence $\mathsf{LouaIVC}$), the circuit $\widetilde{\mathcal{F}}$ 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.

We tackle this problem by finding the balance point between the advantage of utilizing verifier's knowledge and the drawback of handling public inputs in IVC. 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 the encoder and the 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, $\boldsymbol{P}$ and $\boldsymbol{Z}$ are no longer involved the digest computation $\varrho(\mathbb{U}_i, \cdot)$ and $\varrho(\mathbb{U}_{i+1}, \cdot)$ in $\widetilde{\mathcal{F}}$. Instead, the prover only needs to compute their digest once per step in IVC, which occurs in $\mathcal{F}^{\mathsf{Eva}}$.

The soundness is unaffected: the verifier can derive $\hbar_{i+1}$ from $\hbar_i, \boldsymbol{P}, \boldsymbol{Z}$ as well and check the proof against $\hbar_{i+1}$, but the collision resistance of H prevents a malicious prover from providing incorrect $\boldsymbol{P}'$ and $\boldsymbol{Z}'$ that lead to the same $\hbar_{i+1}$.

Proof of Valid Signature. Then we discuss how to prove the validity of σ in $\mathcal{F}^{\mathsf{Eva}}$. 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 5.4.

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, $\mathcal{F}^{\mathsf{Eva}}$ additionally updates the digest h_{i+1} 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 14.

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 Section 5.2). 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 $\mathcal{E}$.

Such a design 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 $\mathsf{param}^{\mathsf{remove}}$.

Circuit 14: $\mathcal{F}^{\mathsf{Eva}}$

```
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 $\boldsymbol{z}_{i+1} := (h_{i+1}, \hbar_{i+1})$
```

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 whether *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.

5.4 Final Protocol

Powered by the IVC scheme Loua that supports lookup arguments in Section 4 and the IVC step circuit in Section 5.3, 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 [78]. 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 $\{0,\dots,255\}$. 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 $\mathsf{LouaIVC}$. 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 5.3, it is still left to compute $\mathsf{H}(h_k, \mathsf{meta})$ and run $\mathsf{Sig}.\mathcal{V}$ on the digest. Since $\mathsf{LouaDecider}.\mathcal{V}$ 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 $\mathbbm{u}_k.\boldsymbol{x}$ from $\mathcal{V}$ 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 $(\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}_k^{\mathsf{cf}}, k), \rho(\mathbbm{u}_k.\overline{Q}))$. Then, the circuit computes $\mathbb{U}_{k+1}^{\mathbb{F}}$ using the field-only operation $\mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}$ (see Appendix B.2), and finally, checks $\mathbb{W}_{k+1}$ against $\mathbb{U}_{k+1}^{\mathbb{F}}$. The final construction of $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ is given in Circuit 15.

Circuit 15: $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$

```
Witness: $h_k, \sigma, \boldsymbol{x}_k, \mathbb{W}_{k+1}, \mathbb{U}_k^{\mathsf{cf}}, \mathbb{W}_k^{\mathsf{cf}}$
Statement: $\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbbm{u}'_k, \mathbb{U}'_k, \overline{T}$
Constant: $\widetilde{\mathsf{CS}}^{\mathsf{Eva}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}), \mathsf{CS}^{\mathsf{cf}} = (\boldsymbol{A}^{\mathsf{cf}}, \boldsymbol{B}^{\mathsf{cf}}, \boldsymbol{C}^{\mathsf{cf}}), \mathsf{ck}^{\mathsf{cf}}$
1 enforce $\mathsf{Sig}.\mathcal{V}(\mathsf{vk}_\Sigma, \sigma, \mathsf{H}(h_k, \mathsf{meta}))$          ▷ Verify $\sigma$
2 Reconstruct $\mathbb{U}_k$ and $\mathbbm{u}_k$:
    $\mathbb{U}_k := \mathbb{U}'_k$
    $\mathbb{U}_k.\boldsymbol{x} := \boldsymbol{x}_k$
    $\mathbbm{u}_k := \mathbbm{u}'_k$
    $\mathbbm{u}_k.\boldsymbol{x} := (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}_k^{\mathsf{cf}}, k), \rho(\mathbbm{u}_k.\overline{Q}))$
3 enforce $r = \rho(\mathbb{U}_k, \mathbbm{u}_k, \overline{T})$          ▷ Check $r$
4 $\mathbb{U}_{k+1}^{\mathbb{F}} := \mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}_\Phi, \mathbb{U}_k^{\mathbb{F}}, \mathbbm{u}_k^{\mathbb{F}}, r)$          ▷ Compute $\mathbb{U}_{k+1}^{\mathbb{F}}$
5 Check $\mathbb{W}_{k+1}$ against $\mathbb{U}_{k+1}^{\mathbb{F}}$:
    Parse $(u, \boldsymbol{x}) := \mathbb{U}_{k+1}^{\mathbb{F}}, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}_{k+1}$
    $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
    enforce $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = u \cdot \boldsymbol{C}\boldsymbol{v} + \boldsymbol{e}$
6 Check $\mathbb{W}_k^{\mathsf{cf}}$ against $\mathbb{U}_k^{\mathsf{cf}}$:
    Parse $(u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}) := \mathbb{U}_k^{\mathsf{cf}}, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}_k^{\mathsf{cf}}$
    $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
    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 $\mathbb{U}_k^{\mathbb{G}}, \mathbb{u}_k^{\mathbb{G}}$ and $\mathbb{U}_{k+1}^{\mathbb{G}}$, 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 $\mathbb{U}_{k+1}^{\mathbb{G}}$ by calling $\mathsf{LouaFS}.\mathcal{V}^{\mathbb{G}}$ (see Appendix B.2) on $\mathbb{U}_k^{\mathbb{G}}, \mathbb{u}_k^{\mathbb{G}}$. The commitments $\overline{Q}, \overline{W}, \overline{E}$ in $\mathbb{U}_{k+1}^{\mathbb{G}}$ 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 $r := \rho(\mathbb{U}_k, \mathbb{u}_k, \overline{T})$, 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 complete Eva protocol in Algorithm 16.

5.5 Security

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

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

6 Implementation and Optimization

We rely on the H.264 reference implementation `JM` [68] to encode and decode videos with the H.264 Main profile. 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[4] in Rust over the BN254/Grumpkin half-pairing cycle of curves. The architecture of our implementation is illustrated in Figure 4, where we highlight the efforts of our own and the improvements to existing work with solid and dashed shapes. In the implementation, we make heavy use of the `arkworks` library [2] for algebraic operations and circuit constructions. We build Loua, our variant of Nova, upon the folding schemes implemented in `sonobe` [75], but we add support for LogUp and introduce various improvements that we will discuss soon. We also provide an alternative implementation of LegoGro16. Unlike the original implementation [45], ours is more flexible and performant: it allows for shared witnesses $(\boldsymbol{v})_{i=0}^{\ell-1}$ with arbitrary length and supports increased parallelism.

In addition, as elaborated below, a bunch of optimizations are applied to maximize the efficiency of the prover.

GPU Acceleration. The prover's cost in our construction is dominated by the computation of commitments $\overline{Q}$, $\overline{W}$, and $\overline{E}$, which involves a multi-scalar multiplication (MSM) operation on $\mathbb{G}$ in Pedersen commitment. Due to the parallelizable nature of MSM, many existing works have investigated the acceleration of MSM on hardware that supports a high degree of parallelism, such as GPUs [63, 87], FPGAs [1], and ASICs [86].

We integrate `icicle` [47]'s GPU implementation of MSM with precomputation into our prover, which provides a 6-7x speedup over the original CPU implementation. While this optimization necessitates extra hardware, GPUs are

[4]The source code can be found here.

Algorithm 16: Eva

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

more accessible and cost-effective than FPGAs and ASICs, especially in our setting where the prover already relies on powerful GPUs for video editing tasks.

We also note that the computation of cross term $\boldsymbol{t} := \boldsymbol{A}\boldsymbol{v}_1 \circ \boldsymbol{B}\boldsymbol{v}_2 + \boldsymbol{A}\boldsymbol{v}_2 \circ \boldsymbol{B}\boldsymbol{v}_1 - u_1 \cdot \boldsymbol{C}\boldsymbol{v}_2 - u_2 \cdot \boldsymbol{C}\boldsymbol{v}_1$ in Loua (inherited from Nova) is another important factor in prover time. Observing that the right hand side essentially requires matrix and vector operations, we can further optimize the prover by leveraging GPU-accelerated linear algebra. Since the R1CS matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are sparse, we implement sparse matrix-vector multiplication (SpMVM) over prime fields in CUDA. These matrices are represented in the compressed sparse row (CSR)

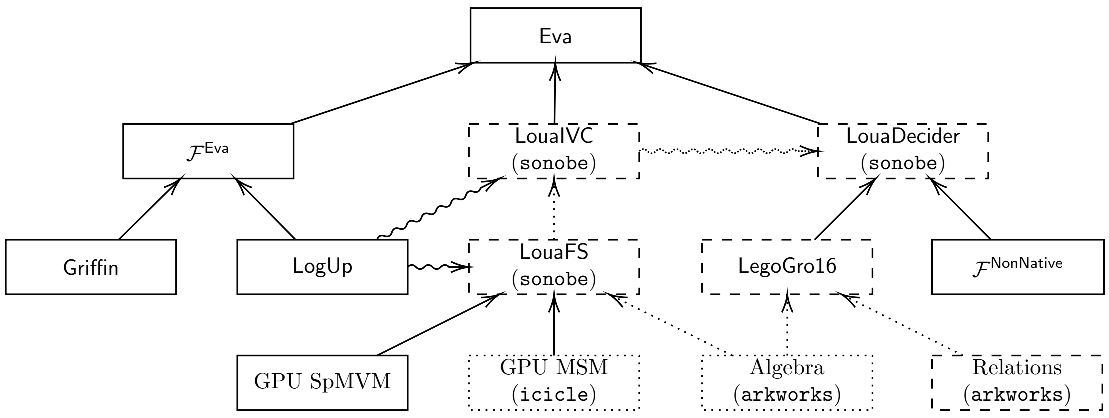


Figure 4: Architecture of Eva's implementation. A box represents a building block, a straight line from X to Y stands for "Y is built upon X", and a waved line from X to Y denotes "Y supports X". Solid shapes are implemented by ourselves from scratch, dashed shapes are our forks of third-party implementations but with significant modifications, and dotted shapes are provided by existing libraries.

format, allowing for better memory access patterns. Further, we adopt a hybrid approach to maximize performance: for each relatively dense row, we assign multiple GPU threads that work together to compute the dot product between the row and the vector, easing the burden of each thread; on the other hand, each relatively sparse row is handled by a single thread to avoid the synchronization costs between threads. With this strategy, workload is nearly evenly distributed across threads, and consequently we improve the prover's time for computing $\boldsymbol{t}$ by 4x to 11x compared to the CPU implementation.

Choice of Hash Function. It is common to use *circuit-friendly* hash functions [36, 10, 35, 37] in SNARKs, among which Poseidon [36] is a popular choice. However, as we need to hash a large amount of data in our circuits for verifying the signature and avoiding complex prediction operations, selecting a more efficient one in our context would greatly reduce the circuit size. We choose Griffin [35] as H, ρ, and ϱ, which, to our knowledge, is the most efficient hash function in terms of the R1CS circuit size, thereby saving up to 50% of constraints compared to Poseidon. Concretely, we instantiate Griffin with degree $d = 5$, state size $t = 24$, and the number of rounds $R = 9$. Note that a large state size is necessary for improving the plain (i.e., bare-metal) performance of Griffin. Otherwise, computing Griffin hashes outside the circuit would be slower than Poseidon due to the high degree exponentiation $x^{1/d}$.

Amortizing Constraints for Folding Verification. Compared with recursive SNARKs [5], folding-based IVC reduces the prover's overhead by avoiding the in-circuit verification of a SNARK proof. However, in addition to the evaluation of $\mathcal{F}$, the prover still needs to prove the folding verification algorithm $\mathsf{NIFS}.\mathcal{V}$ in each step circuit $\widetilde{\mathcal{F}}$. Existing techniques, such as cycle of curves [72] and CycleFold [56], makes the folding verification circuit practically small (less than 10^5 constraints), but this additional cost is prohibitive for a small $\mathcal{F}$.

For instance, with LouaFS as the folding scheme, we analyze two components of Eva's augmented circuit $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$: the inner $\mathcal{F}^{\mathsf{Eva}}$ circuit that handles b macroblocks per step, and the gadget $\mathsf{LouaFS}.\mathcal{V}$ for folding verification. When $b = 1$, the former has ~ 3500 constraints, while the latter requires ~ 67000 constraints, which is a 19x increase in the prover's cost.

To minimize such overhead, we amortize the constraints for $\mathsf{LouaFS}.\mathcal{V}$ by processing multiple macroblocks in batch in each step, so that the prover only needs to prove $\mathsf{LouaFS}.\mathcal{V}$ once for every b macroblocks. The larger b is, the more the prover can save on the cost of $\mathsf{LouaFS}.\mathcal{V}$. On the downside, a large b would increase the circuit size of $\mathcal{F}^{\mathsf{Eva}}$, imposing a higher memory requirement on the prover. As a trade-off between time and space, we set the batch size to $b = 256$ in our implementation, thereby reducing the cost of $\mathsf{LouaFS}.\mathcal{V}$ to ~ 260 constraints per macroblock, while $\mathcal{F}^{\mathsf{Eva}}$ has a reasonable size (~ 760000 constraints).

Parallel Circuit Synthesis. Another efficiency bottleneck in our implementation is the synthesis (*i.e.*, creation) of the step circuit. As discussed above, the prover now needs to process $b = 256$ macroblocks per step. Hence, the $\mathcal{F}^{\mathsf{Eva}}$ circuit essentially consists of b copies of the logic for processing a single macroblock, which are sequentially converted into constraints when synthesizing the circuit in `arkworks`. It is natural to ask whether we can parallelize the processing of these b macroblocks, which would significantly reduce the time for circuit generation. Unfortunately, `arkworks` does not support such parallelism, since a constraint in general may depend on previously computed variables, although in our case, the constraints for each macroblock are independent of each other.

As a workaround, we 1) synthesize a *dummy circuit* for a dummy macroblock, 2) create b *partial circuits*, each of which handles one macroblock in the video, and then 3) merge the the partial circuits into the *final circuit* by concatenating the variables and constraints. The indices of variables and constraints in each partial circuit are offset by the number of variables and constraints in the dummy circuit, in order to avoid overlapping variables and constraints in the final circuit. In this way, we can parallelize 2), the most time-consuming step, without breaking the internal sequential dependencies of variables and constraints in partial circuits.

Efficient Non-Native Field Operations. In $\mathcal{F}^{\mathsf{Decider}}$ (see Appendix B.3), we need to check $\boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v} \equiv u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e} \pmod q$ in a non-native field $\mathbb{F}_q$. To this end, we can apply the in-circuit big integer arithmetic proposed in [55], which allows for efficient operations with arbitrary precision. The high-level idea behind [55] is to represent a big integer as a vector of *limbs* in the native field, and then perform limb-wise arithmetic operations. Note that since the native field cannot contain arbitrarily large limbs, we need to *align* the bitwidth of each limb after a certain number of operations, which is done by performing the expensive bit decomposition operation. Thus, when checking the equality of two big integers that are not necessarily aligned, the circuit size would become very large if we naively decompose and align the limbs before the actual comparison. While [55] decreases the number of bit decompositions in equality checks, they are still the most costly operation in the circuit.

Utilizing [55], a straightforward approach to emulation of field operations in $\mathbb{F}_q$ is to perform every operation modulo q. For instance, to multiply two non-native field elements a, b, we need to compute $c \bmod q$ after performing the big integer multiplication $c := a \cdot b$, so that the resulting big integer c is in $\mathbb{F}_q$. The in-circuit modulo operation can be implemented by asking the prover to provide the quotient s and the remainder r as hints, whose validity is checked by enforcing $c = sq + r$, $0 \leq s < c$, and $0 \leq r < q$. Nevertheless, due to the complexity of the equality check, the final circuit relying on modulo operations would have $\sim 4 \times 10^7$ constraints[5] and require more than 200 GB of memory.

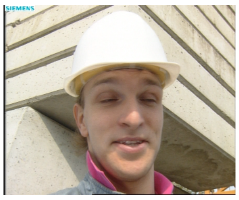
To further improve the efficiency, we defer the modulo operation to the end of the circuit. That is, we avoid performing modulo operations during the computation of $LHS \coloneqq \boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v}$ and $RHS \coloneqq u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e}$. All intermediate results are treated as big integers, and LHS and RHS are converted to elements in $\mathbb{F}_q$ only before the final equality check.

In addition, we observe that it is unnecessary to compute both $LHS \bmod q$ and $RHS \bmod q$ when checking $LHS \equiv RHS \pmod{q}$. Instead, we can compute $LHS - RHS$ as a big integer and check whether $LHS - RHS$ is a multiple of q.

Now, we successfully get rid of all modulo operations in the circuit. Although the intermediate values during the computation of LHS and RHS contain more limbs, resulting in more expensive multiplication operations, the overall cost is still much lower than the naive approach thanks to the elimination of modulo operations. In fact, the number of constraints for checking $\boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v} \equiv u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e} \pmod{q}$ is reduced to $\sim 2.5 \times 10^6$, which is a 16x improvement over the naive approach.

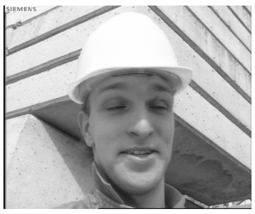
7 Evaluation

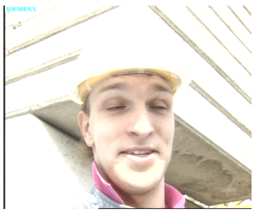
To evaluate our implementation of Eva, we compile it with multi-threading and AVX2 enabled, and run it on a consumer-grade PC equipped with an Intel Core i9-12900K CPU (16 cores, 24 threads) with 64 GB of RAM and an NVIDIA GeForce RTX 3080 GPU with 12 GB of VRAM.

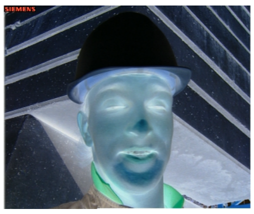

(a) foreman.yuv

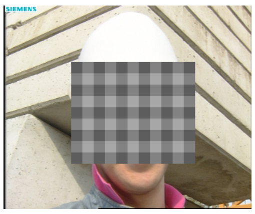
(b) bunny.yuv

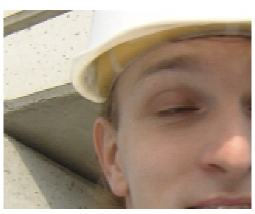
Figure 5: Preview of original videos in the test dataset


(a) Δ_{gray}


(b) Δ_{bright}
$\mathsf{param}^{\mathsf{bright}} = 416/256$


(c) Δ_{inv}


(d) Δ_{mask}
Mask of size 176×144


(e) Δ_{remove} (cropping)
$352 \times 288 \to 160 \times 128$

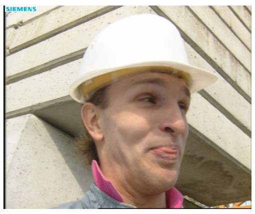

(f) Δ_{remove} (cutting)
256 frames $\to$ 128 frames

Figure 6: Preview of videos edited from "foreman.yuv"

For testing purposes, we utilize two raw video files that are widely used for video codec benchmarking, as shown in Figure 5. The first video, "foreman.yuv," contains 256 frames of size 352×288, which is used to demonstrate Eva's capability

[5]See https://hackmd.io/x821TH5oTcKE3uPHniuefw.

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 6. 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).

Table 2: Breakdown of the number of R1CS constraints in $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$ 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	0	0	0	0	0	
			$\mathcal{F}^{\mathsf{Trans}}$	0	0	0	0	0	0	
			$\mathcal{F}^{\mathsf{Quant}}$ Y	1328	1328	1328	1328	1328	1328	
			$\mathcal{F}^{\mathsf{Quant}}$ U	320	0	320	320	320	320	
			$\mathcal{F}^{\mathsf{Quant}}$ V	320	0	320	320	320	320	
			$\mathsf{H}(\boldsymbol{X})$	306	306	306	306	306	306	
			$\mathsf{H}(\boldsymbol{P}, \boldsymbol{Z}, \mathsf{param})$	612	612	612	612	918	613	
			$\mathsf{H}(h_i, \cdots)$	5508	5508	5508	5508	5508	5508	
			$\mathsf{H}(\hbar_i, \cdots)$	5508	5508	5508	5508	5508	5511	
			Subtotal	760584	596744	1022728	760584	1035528	859403	
Augmentation			Create variables	6865	6865	6865	6865	6865	6865	
			Fold $\mathbb{u}_i^{\mathbb{F}}$ into $\mathbb{U}_i^{\mathbb{F}}$	13361	13361	13361	13361	13361	13361	
			Fold $\mathbb{u}_i^{\mathsf{cf}}$ into $\mathbb{U}_i^{\mathsf{cf}}$	45108	45108	45108	45108	45108	45108	
			Check lookup identity	594177	430337	790785	594177	692481	594177	
			Compute outputs	9117	9117	9117	9117	9117	9117	
			Subtotal	668628	504788	865236	668628	766932	668628	
			Total	1429212	1101532	1887964	1429212	1802460	1528031	

Table 3: Breakdown of the number of R1CS constraints in $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ with $b = 256$.

Subroutine \ Editing Op.	Δ_{id}	Δ_{gray}	Δ_{bright}	Δ_{inv}	Δ_{mask}	Δ_{remove}
Create variables	695968	695968	695968	695968	695968	695968
Verify signature	4590	4590	4590	4590	4590	4590
Reconstruct instances	8173	8173	8173	8173	8173	8173
Check r	5186	5186	5186	5186	5186	5186
Fold $\mathbb{u}_k^{\mathbb{F}}$ into $\mathbb{U}_k^{\mathbb{F}}$	3	3	3	3	3	3
Check $\widetilde{\mathsf{CS}}^{\mathsf{Eva}}$ satisfiability	2705351	2082759	3557319	2705351	3353543	2902732
Check $\mathsf{CS}^{\mathsf{cf}}$ satisfiability	2575044	2575044	2575044	2575044	2575044	2575044
Verify commitments in $\mathbb{U}_k^{\mathsf{cf}}$	3544918	3544918	3544918	3544918	3544918	3544918
Total	9539233	8916641	10391201	9539233	10187425	9736614

Circuit efficiency. An important metric for evaluating the performance of a protocol based on general-purpose SNARKs is the efficiency of the arithmetic circuits. We measure the circuit efficiency of Eva by the number of R1CS constraints in our augmented step circuit $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$ and decider circuit $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$.

We first report the number of constraints in $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$ in Table 2, where Δ_{id} is an identity function (i.e., no edits are performed). We can observe that the sizes of $\mathcal{F}^{\mathsf{Eva}}$ with all editing operations are between 600K and 1M constraints. When augmenting the circuit for the use of IVC, the additional cost is dominated by the check of lookup identity (500K to 860K constraints), as our $\mathcal{F}^{\mathsf{Eva}}$ makes heavy use of lookup tables for efficient in-circuit editing and encoding. In fact, if we replace the lookup arguments with bit decompositions, the constraints for the lookup identity check would increase by approximately eightfold (since our $\boldsymbol{\tau}$ contains 8-bit entries) in $\mathcal{F}^{\mathsf{Eva}}$, resulting in nearly an order of magnitude increase in the overall circuit size.

Next, we also list the number of constraints in our decider circuit $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$, as shown in Table 3. The size of $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ also depends on the editing operation, because we need to check the satisfiability of $\mathbb{W}_k$ and $\mathbb{U}_k^{\mathbb{F}}$ against $\widetilde{\mathsf{CS}}^{\mathsf{Eva}}$, which has different dimensions for different Δ. The dominant parts of the circuit are the checks of relaxed R1CS satisfiability and commitment verification, each of which introduces $\sim$ 3M constraints, resulting in a total of $\sim$ 10M constraints for $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$.

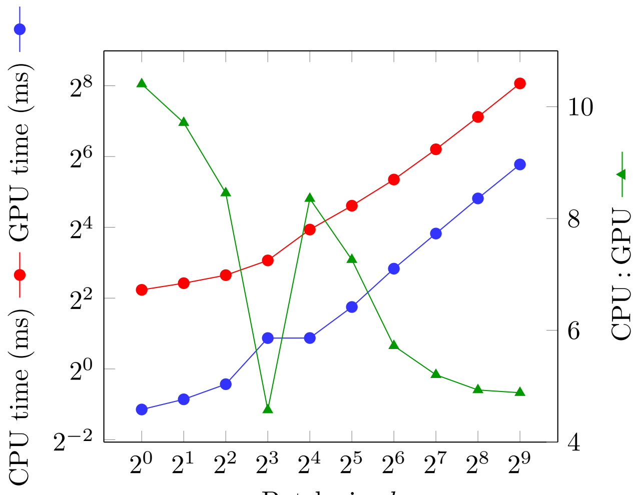


Figure 7: Running time of cross term computation on CPU and GPU w.r.t. the batch size b.

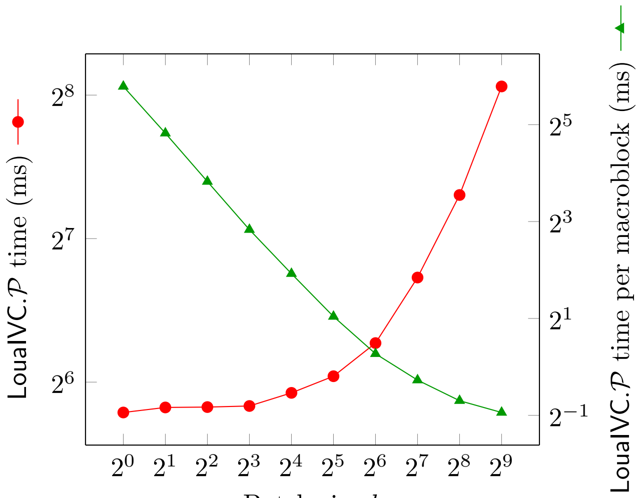


Figure 8: Running time of LoualVC.$\mathcal{P}$ w.r.t. the batch size b, where the editing operation is fixed to $\Delta = \Delta_{\mathsf{id}}$.

Microbenchmarks. We conduct microbenchmarks to evaluate the performance of the prover in Eva.

First, we study the impact of GPU acceleration on the prover's performance by measuring the time for computing Nova's cross term $\boldsymbol{t}$ on CPU and GPU. Here, R1CS matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are generated from the augmented step circuit $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$ for Eva, with batch size b ranging from 2^0 to 2^9. The results are shown in Figure 7, from which we can observe that the GPU outperforms the CPU by a factor of $4 \sim 11$.

Second, we evaluate the running time of LoualVC.$\mathcal{P}$ with respect to the batch size b. As we can see in Figure 8, when the batch size b is small, doubling b nearly halves the average running time of LoualVC.$\mathcal{P}$ for each macroblock. This is because for a small b, the dominant part of the augmented step circuit is still the in-circuit folding verification, thereby demonstrating the effectiveness of amortizing the constraints for LouaFS.$\mathcal{V}$ through batching.

Finally, we fix the batch size $b = 256$ and study how the editing operation Δ

affects the prover performance in Eva. We report the running time and RAM usage of $\mathsf{LouaIVC}.\mathcal{P}$ and $\mathsf{LouaDecider}.\mathcal{P}$ in Table 4. In order to illustrate how the number of constraints affects prover time and RAM usage, we also include the sizes of $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$ and $\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}$ summarized in Table 2 and Table 3. Across different operations, the running time of $\mathsf{LouaIVC}.\mathcal{P}$ ranges from 145 to 206 ms, and the peak RAM usage varies from 7 to 11 GB. $\mathsf{LouaDecider}.\mathcal{P}$ 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 $\mathsf{Eva}.\mathcal{P}$.

Table 4: Benchmarking results of $\mathsf{LouaIVC}.\mathcal{P}$ and $\mathsf{LouaDecider}.\mathcal{P}$ for different editing operation Δ with $b = 256$.

		Δ_{id}	Δ_{gray}	Δ_{bright}	Δ_{inv}	Δ_{mask}	Δ_{remove}
$\mathsf{LouaIVC}.\mathcal{P}$	$\lvert\widetilde{\mathcal{F}}^{\mathsf{Eva}}\rvert$	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
$\mathsf{LouaIVC}.\mathcal{P}$	$\lvert\mathcal{F}^{\mathsf{Decider}_{\mathsf{Eva}}}\rvert$	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 5: 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 IVC steps (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

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 5.

In $\mathsf{Eva}.\mathcal{P}$, 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. Additional 65 $\sim$ 80 seconds are needed to make the proof fully succinct and zero-knowledge by running $\mathsf{ZKCP}.\mathcal{P}$ 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 $\mathsf{ZKCP}.\mathcal{P}$ step.

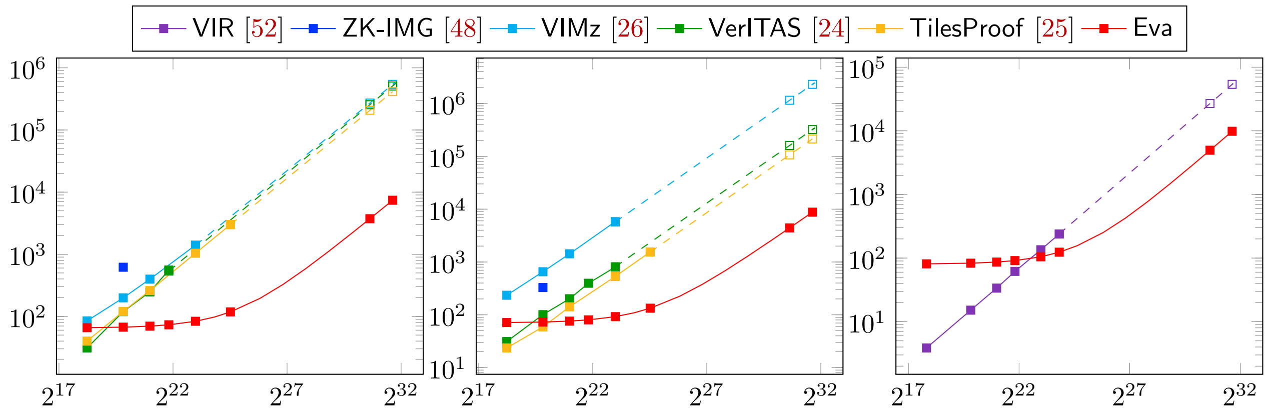
From the results, we conclude that the total IVC time scales linearly with the video size. Specifically, the IVC time for any video of size $M \times N \times L$ can be estimated by computing the number of IVC steps $k = (M/16 \times N/16 \times L)/b$ and scaling the single step time of $\mathsf{LouaIVC}.\mathcal{P}$ in Table 4. 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 5; 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 [52, 48, 26, 24, 25], focusing specifically on the prover time. PhotoProof [69] is not included in the comparison, as it only supports tiny images of size up to 128 $\times$ 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.



(a) Grayscale conversion **(b)** Cropping **(c)** Masking

Figure 9: Comparison of prover time (y-axis, in seconds) across different protocols w.r.t. the number of pixels (x-axis).

For protocols that support arbitrary editing operations [48, 26, 24, 25], we select two representative operations for comparison: grayscale conversion, representing color manipulations, and cropping, representing spatial modifications. The target resolution for the cropping operation is set to 640 $\times$ 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 $\rightarrow$ SD)" on HD (1280 $\times$ 720) images reported in [48, 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 time would likely be slower on our machine. All remaining protocols are evaluated on the same machine as Eva.

The results of prover time are given in Figure 9a and Figure 9b, respectively, 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 factor in our prover, allowing the advantages of our efficient $\mathsf{IVC}.\mathcal{P}$ 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 the editing operation Δ.

For VIR [52], 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 9c. 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".

8 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 [17], π is very small and only require 2 pairings for verification: one for Groth16 (3 Miller loops, 1 final exponentiation), and the other for the zkSNARK for Linear Subspaces [51] (5 Miller loops, 1 final exponentiation). According to EIP-1108 [19], we estimate that the cost of verifying π on EVM would be close to $34000 \times (3 + 5) + 45000 \times (1 + 1) = 362000$ gas, or equivalently ~ 16 USD as of August 2024.

In comparison, although Dziembowski et al. claim that VIMz [26] 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 [64] for verifying Spartan proofs on EVM, requiring ~ 200M gas[6] or ~ 9000 USD.

[6]For details, see https://github.com/lurk-lab/solidity-verifier/issues/29.

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.

However, $\mathcal{R}$ is usually resource-constrained and only allows lossily encoded videos in practice. In this case, $\mathcal{P}$ 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 not exactly the same as 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, since $\mathcal{R}$ 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] Kaveh Aasaraai et al. “FPGA acceleration of multi-scalar multiplication: Cyclonemsm”. In: *Cryptology ePrint Archive* (2022).

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

[3] *AV1 Bitstream & Decoding Process Specification.* 1st ed. Alliance for Open Media. 2019. URL: https://aomediacodec.github.io/av1-spec/av1-spec.pdf.

[4] Eli Ben-Sasson et al. “Fast reed-solomon interactive oracle proofs of proximity”. In: *45th international colloquium on automata, languages, and programming (icalp 2018).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[5] Eli Ben-Sasson et al. “Scalable Zero Knowledge via Cycles of Elliptic Curves”. In: *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34.* Springer. 2014, pp. 276–294.

[6] Eli Ben-Sasson et al. "SNARKs for C: Verifying program executions succinctly and in zero knowledge". In: *Annual cryptology conference*. Springer. 2013, pp. 90–108.

[7] Nir Bitansky et al. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again". In: *Proceedings of the 3rd innovations in theoretical computer science conference*. 2012, pp. 326–349.

[8] Jonathan Bootle et al. "Arya: Nearly linear-time zero-knowledge proofs for correct program execution". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 595–626.

[9] Gautam Botrel et al. *gnark is a fast zk-SNARK library that offers a high-level API to design circuits*. https://github.com/Consensys/gnark. 2023.

[10] Clémence Bouvier et al. "New design techniques for efficient arithmetization-oriented hash functions: anemoi permutations and jive compression mode". In: *Annual International Cryptology Conference*. Springer. 2023, pp. 507–539.

[11] 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*. Springer. 2023, pp. 77–110.

[12] Benedikt Bünz and Jessica Chen. "Proofs for Deep Thought: Accumulation for large memories and deterministic computations". In: *Cryptology ePrint Archive* (2024).

[13] 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*. Springer. 2020, pp. 677–706.

[14] Benedikt Bünz et al. "Bulletproofs: Short proofs for confidential transactions and more". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 315–334.

[15] Benedikt Bünz et al. "Proof-Carrying Data from Accumulation Schemes". In: (2020).

[16] Benedikt Bünz et al. "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*. Springer. 2021, pp. 681–710.

[17] 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*. 2019, pp. 2075–2092.

[18] Matteo Campanelli et al. “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*. Springer. 2021, pp. 3–33.

[19] Antonio Salazar Cardozo and Zachary Williamson. *EIP-1108: Reduce alt_bn128 precompile gas costs*. https://eips.ethereum.org/EIPS/eip-1108. May 21, 2018.

[20] 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*. Springer. 2013, pp. 55–72.

[21] Alessandro Chiesa and Eran Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards.” In: *Innovations in Computer Science - ICS 2010*. Vol. 10. 2010, pp. 310–331.

[22] *Content Credentials: C2PA Technical Specification*. 2.0. Coalition for Content Provenance and Authenticity. 2024. URL: https://c2pa.org/specifications/specifications/2.0/specs/C2PA_Specification.html.

[23] Baris Coskun, Bulent Sankur, and Nasir Memon. “Spatio–temporal transform based video hashing”. In: *ieee Transactions on Multimedia* 8.6 (2006), pp. 1190–1208.

[24] Trisha Datta, Binyi Chen, and Dan Boneh. “VerITAS: Verifying Image Transformations at Scale”. In: *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2024, pp. 97–97.

[25] Pierpaolo Della Monica et al. “Trust Nobody: Privacy-Preserving Proofs for Edited Photos with Your Laptop”. In: *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2024, pp. 14–14.

[26] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. “VIMz: Private proofs of image manipulation using folding-based zkSNARKs”. In: *Proceedings on Privacy Enhancing Technologies* (2025).

[27] Liam Eagen and Ariel Gabizon. “ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances”. In: *Cryptology ePrint Archive* (2023).

[28] Jens Ernstberger et al. “ Zero-Knowledge Location Privacy via Accurate Floating-Point SNARKs ”. In: *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, 2025, pp. 57–57.

[29] Zach Evans et al. “Fast Timing-Conditioned Latent Audio Diffusion”. In: *Forty-first International Conference on Machine Learning*. 2024.

[30] Mahmoud E Farfoura et al. “Low complexity semi-fragile watermarking scheme for H. 264/AVC authentication”. In: *Multimedia Tools and Applications* 75 (2016), pp. 7465–7493.

[31] 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*. Springer. 1986, pp. 186–194.

[32] Ariel Gabizon and Zachary J Williamson. "plookup: A simplified polynomial protocol for lookup tables". In: *Cryptology ePrint Archive* (2020).

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

[34] Rosario Gennaro et al. "Quadratic span programs and succinct NIZKs without PCPs". In: *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer. 2013, pp. 626–645.

[35] Lorenzo Grassi et al. "Horst meets fluid-spn: Griffin for zero-knowledge applications". In: *Annual International Cryptology Conference*. Springer. 2023, pp. 573–606.

[36] Lorenzo Grassi et al. "Poseidon: A new hash function for Zero-Knowledge proof systems". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 519–535.

[37] Lorenzo Grassi et al. "Reinforced concrete: a fast hash function for verifiable computation". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1323–1335.

[38] Matthew Groh et al. "Deepfake detection by human crowds, machines, and machine-informed crowds". In: *Proceedings of the National Academy of Sciences* 119.1 (2022), e2110013119.

[39] 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*. Springer. 2016, pp. 305–326.

[40] *H.264: Advanced video coding for generic audiovisual services*. 14th ed. ITU Telecommunication Standardization Sector. 2021. URL: https://www.itu.int/rec/T-REC-H.264-202108-I/en.

[41] *H.265: High efficiency video coding*. 9th ed. ITU Telecommunication Standardization Sector. 2023. URL: https://www.itu.int/rec/T-REC-H.265-202309-I/en.

[42] Ulrich Haböck. "Multivariate lookups based on logarithmic derivatives". In: *Cryptology ePrint Archive* (2022).

[43] Alexandros Haliassos et al. "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*. 2021, pp. 5039–5049.

[44] Qingying Hao et al. "It's not what it looks like: Manipulating perceptual hashing based applications". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 69–85.

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

[46] Shehzeen Hussain et al. “Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples”. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 3348–3357.

[47] Ingonyama. *icicle: a GPU Library for Zero-Knowledge Acceleration*. https://github.com/ingonyama-zk/icicle. 2023.

[48] Daniel Kang et al. “ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation”. In: *arXiv preprint arXiv:2211.04775* (2022).

[49] 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*. Springer. 2010, pp. 177–194.

[50] Fouad Khelifi and Ahmed Bouridane. “Perceptual video hashing for content identification and authentication”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.1 (2017), pp. 50–67.

[51] Eike Kiltz and Hoeteck Wee. “Quasi-adaptive NIZK for linear subspaces revisited”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 101–128.

[52] Hankyung Ko et al. “Efficient verifiable image redacting based on zk-SNARKs”. In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 2021, pp. 213–226.

[53] Dan Kondratyuk et al. “VideoPoet: A Large Language Model for Zero-Shot Video Generation”. In: *Forty-first International Conference on Machine Learning*. 2024.

[54] Pavel Korshunov and Sébastien Marcel. “Subjective and objective evaluation of deepfake videos”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 2510–2514.

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

[56] Abhiram Kothapalli and Srinath Setty. “CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves”. In: *Cryptology ePrint Archive* (2023).

[57] Abhiram Kothapalli and Srinath Setty. “HyperNova: Recursive arguments for customizable constraint systems”. In: *Annual International Cryptology Conference*. Springer. 2024, pp. 345–379.

[58] Abhiram Kothapalli and Srinath Setty. “NeutronNova: Folding everything that reduces to zero-check”. In: *Cryptology ePrint Archive* (2024).

[59] Abhiram Kothapalli and Srinath Setty. “SuperNova: Proving universal machine executions without universal circuits”. In: *Cryptology ePrint Archive* (2022).

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

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

[62] Yuezun Li et al. “Celeb-df: A large-scale challenging dataset for deepfake forensics”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 3207–3216.

[63] Tao Lu et al. “cuZK: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on GPUs”. In: *Cryptology ePrint Archive* (2022).

[64] Lurk Lab. *solidity-verifier: Solidity implementation of Nova proving system verifier*. https://github.com/lurk-lab/solidity-verifier. 2023.

[65] Iacopo Masi et al. “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*. Springer. 2020, pp. 667–684.

[66] Microsoft. *PhotoDNA*. https://www.microsoft.com/en-us/photodna.

[67] 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. Jan. 12, 2022.

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

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

[70] Paarth Neekhara et al. “Adversarial threats to deepfake detection: A practical perspective”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 923–932.

[71] Paarth Neekhara et al. “FaceSigns: Semi-Fragile Watermarks for Media Authentication”. In: *ACM Transactions on Multimedia Computing, Communications and Applications* (2024).

[72] Wilson Nguyen, Dan Boneh, and Srinath Setty. “Revisiting the Nova proof system on a cycle of curves”. In: *Cryptology ePrint Archive* (2023).

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

[74] Polygon Zero. *Plonky2: a SNARK implementation based on techniques from PLONK and FRI*. https://github.com/0xPolygonZero/plonky2. 2021.

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

[76] Jonathan Prokos et al. “Squint hard enough: Attacking perceptual hashing with adversarial machine learning”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 211–228.

[77] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[78] Claus-Peter Schnorr. “Efficient identification and signatures for smart cards”. In: *Advances in Cryptology—CRYPTO’89 Proceedings 9*. Springer. 1990, pp. 239–252.

[79] Srinath Setty. “Spartan: Efficient and general-purpose zkSNARKs without trusted setup”. In: *Annual International Cryptology Conference*. Springer. 2020, pp. 704–737.

[80] 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*. Springer. 2024, pp. 180–209.

[81] Rashid Tahir et al. “Seeing is believing: Exploring perceptual differences in deepfake videos”. In: *Proceedings of the 2021 CHI conference on human factors in computing systems*. 2021, pp. 1–16.

[82] 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. Sept. 18, 2022.

[83] 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*. Springer. 2008, pp. 1–18.

[84] Arantxa Zapico et al. “Caulk: Lookup arguments in sublinear time”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 3121–3134.

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

[86] Ye Zhang et al. “Pipezk: Accelerating zero-knowledge proof with a pipelined architecture”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 416–428.

[87] Xudong Zhu et al. “Elastic MSM: A Fast, Elastic and Modular Preprocessing Technique for Multi-Scalar Multiplication Algorithm on GPUs”. In: *Cryptology ePrint Archive* (2024).

Appendix A Additional Related Work

In the following, we provide a detailed overview of related work. First, we examine cryptographic protocols for image authentication since they have a close relationship to video authentication and rely heavily on cryptographic proofs as well. Second, we discuss non-cryptographic methods for authenticating genuine videos and detecting fake videos.

Image authentication based on cryptographic proofs. The advance of succinct proof systems like zkSNARKs has made it possible to prove statements previously deemed infeasible, enabling the development of cryptographic protocols for image authentication. The pioneering work in this direction, PhotoProof [69], uses Proof-Carrying Data (PCD) [21] to prove the authenticity of edited images. Specifically, the proof demonstrates that the edited image m' is derived from the original image m, and the signature on the hash of m is valid. Due to the high computational cost of the proof generation, the authors only evaluate PhotoProof on images with a maximum resolution of 128×128 pixel. In [52], Ko et al. propose VIR, a verifiable image redacting protocol based on CP-SNARKs [17]. By focusing solely on the operation of redacting images (masking secret parts with black tiles), VIR significantly reduces prover time ($\sim$ 300x smaller) and supports much larger images, up to 3840×2160 resolution. Built upon a more efficient proof system `halo2` [85], ZK-IMG [48] also has $\sim$ 100x faster prover than PhotoProof, while maintaining support for arbitrary editing operations.

Concurrent with our work, Dziembowski et al. introduce VIMz [26], Datta et al. propose VerITAS [24], and Della Monica et al. present TilesProof [25], which share several common ideas with Eva. For instance, VIMz also employs folding schemes to reduce prover RAM costs; VerITAS utilizes lookup arguments as well to improve prover time; similar to our approach, VIMz and TilesProof save prover RAM by splitting the image into blocks (rows in VIMz and tiles in TilesProof) and proving each block separately. However, alongside these general techniques, Eva incorporates a range of tailored optimizations to minimize prover time, resulting in better performance than all these protocols. In terms of image size, all concurrent schemes support high-resolution images, among which VerITAS and TilesProof even showcases proof generation for images with $\sim$ 30M pixels. VerITAS makes this practical thanks to its custom proof system for proving pre-image of lattice-based hash functions, while TilesProof addresses this by proving the transformation on each small tile separately.

Considering that videos can be seen as a generalization of images, we provide a comparison between Eva and cryptographic image authentication protocols in Table 1, in terms of supported format and compression modes, allowed editing operations, prover time and RAM usage, proof size, and maximum dimensions of the input data.

Performance. We compare the prover time complexity w.r.t. P, the number of pixels of the image or video. For TilesProof, an additional adjustable parameter T is introduced to denote the number of tiles. Furthermore, we evaluate the per-pixel prover time of all protocols for an editing operation with average complexity. If source code is available, we ran experiments ourselves and provide the concrete prover time on our machine. Otherwise, we refer to the authors' evaluation.

As discussed above, the prover performance in PhotoProof is suboptimal. With a time complexity of $O(P^3 \log P)$, PhotoProof takes $\sim$ 18676 µs/px based on the authors' evaluation on a lower-spec machine compared to ours. In contrast, VIR demonstrates significant improvement in prover time due to its use of dedicated proof systems for specific editing operations, resulting in a per-pixel prover time of $\sim$ 16 µs and $O(P \log P)$ time complexity. While ZK-IMG also achieves performance gain over PhotoProof, the proof generation still takes a considerable amount of time, especially when proof of hash is involved ($>$ 355 µs, where $>$ is used because this was evaluated on a very high-performance server). Compared to ZK-IMG, VIMz further reduces prover time by $2 \sim 3$x, averaging around 167 µs/px with a linear complexity. Similarly, VerITAS offers fast prover performance at about 95 µs/px by optimizing the time for proof of hash, although its complexity is still $O(P \log P)$ due to polynomial interpolation. By proving the transformation on each tile separately, TilesProof achieves a per-pixel prover time of 62.308 µs. Since TilesProof uses Groth16 [39] as the proof system, which relies on NTT to convert R1CS witnesses to QAP witnesses, the prover time for each tile is $O(P/T \log(P/T))$ and the total prover time is $O(P \log(P/T))$. Eva, due to the combination of customized folding scheme, tailored circuit design, and various optimizations in our implementation, achieves optimal time complexity ($O(P)$) and the fastest prover time (~ 5 µs/px) among all the protocols.

Thanks to the macroblock-based structure of video encoding, Eva only needs to handle a fixed number of macroblocks in each incremental step of IVC, controlled by a constant batch size b. In comparison, most of prior works [69, 52, 48, 24] require RAM proportional to the image size. This is because they either load the entire image into the arithmetic circuit or, in the case of VIR, use a structured reference string srs containing commitment keys for the entire image. While VIMz achieves lower memory costs through folding schemes, its memory usage remains linear in the image width N, because of its row-by-row proof generation process. The memory required by TilesProof depends on the number of tiles T, and for $T = P/c$ with constant c, the memory usage could become a constant $O(c)$, at the cost of proof size linear in P.

Regarding proof size, both [69, 52] generate proofs of constant size (2.67 KB and 223 B, respectively). This is also the case for Eva, which produces proofs of size 448 B. In contrast, the proofs in ZK-IMG [48] have a size of $O(\log P)$. This is due to ZK-IMG utilizing `Halo2` [85], which is based on the inner-product argument from [15] that generates proofs of size $O(\log n)$ in the number of constraints n. Here, n is linear in the number of pixels P, according to the circuit design described in [48, Section 7]. Meanwhile, both VIMz and VerITAS produce proofs of size $O(\log^2 n)$, because the former leverages Spartan [79] as the decider, and the latter is powered by `Plonky2` [74] with FRI [4] as the commitment scheme. For VIMz, n is linear in the image width N, while for VerITAS, n is proportional to P. Concretely, the proofs from ZK-IMG, VIMz, and VerITAS are at least 20 times larger than those of Eva. As mentioned earlier, TilesProof makes a trade-off between proof size and memory usage. While it is able to generate constant-size proofs, the RAM consumption will be linear in P in this case.

Functionality. Unlike existing protocols that only support lossless editing operations on images, Eva is the first to provide authenticity for lossily encoded videos, addressing challenges of complex compression algorithms and large data sizes.

In terms of supported editing operations, the design of VIR only allows redaction, which can be seen as a special case of masking, whereas protocols based on general-purpose zkSNARKs [69, 48, 26, 24, 25] support arbitrary editing operations, and this is also the case for Eva. However, previous protocols have their own limitations.

- In PhotoProof, only predefined permissible edits are considered legitimate transformations, while all others are rejected. In contrast, Eva does not require predefined permissible edits. The prover can demonstrate any editing operation, and it is up to the verifier to decide if they are happy with the claimed transformation.

- When performing cropping operations, VIMz requires cropped images to have fixed dimensions, and we observe that VerITAS and TilesProof also have similar restrictions in their implementations. This is not desirable in practice, as trusted setup is required for each possible crop size. Eva, on the other hand, supports cropping sizes parameterized by a dynamic value (see Sections 5.2.2 and 5.3 for details), thereby eliminating the need for multiple trusted setups for different sizes.

- Due to the design of TilesProof, only transformations that can be applied locally to each tile are supported, while the authors regard global transformations like rotation and flipping as out of scope. Although the prover in Eva also processes the video in a block-wise manner, it is able to handle global transformations by leveraging vector commitment schemes, as explained in Section 5.2.3.

Furthermore, due to its constant RAM consumption, Eva supports videos with unlimited resolution and frame count, while the other protocols cannot achieve infinite resolution, because their RAM usage scales with image dimensions, but the prover only has bounded RAM in practice.

Prior video authentication protocols. In video authentication, the prover generates authentication information for a claimed video, which can later be verified either publicly or privately. Existing work regarding video authentication follows two technique routes. The first is based on robust hash [23, 50] (sometimes referred to as perceptual hash [66, 50]), a digest extraction algorithm whose output is robust against benign transformations (e.g., resizing, (re-)encoding, cropping) but fragile to malicious manipulations (e.g., object replacement). After the robust hash is extracted, the prover generates authentication information by feeding the resulting hash value to, e.g., signing and watermarking.

However, it is challenging to define transformations that achieve the balance between robustness and fragility. Consider a toy example: if fragility is determined by the number of altered pixels, then minor but malicious edits (e.g., changing a number on a banknote) might pass as acceptable, while significant but benign edits (e.g., cropping to remove a person) might be rejected. In fact, these protocols experience non-negligible false positive or false negative rates [30, 71, 23, 50] and active attackers can bypass some of these mechanisms [44, 76]. Additionally, having predefined legal transformations may not be practical, because whether a transformation is benign or malicious can be subjective and context-dependent.

The second is adopted by *Coalition for Content Provenance and Authenticity (C2PA)* [22], an industry standard for multimedia authentication based on digital signatures. C2PA requires that recording devices, such as mobile phones or cameras, have built-in signing keys certified by the device manufacturer. When multimedia content is recorded, the device generates a signature for both the content and its metadata, which may include thumbnails, capture date, location, etc. The multimedia content can later be edited by trusted editing software, which also has signing keys embedded. Analogously, the editing software signs the processed content, the metadata, and the editing operations performed. Upon publishing the processed content along with these signatures, the verifier (e.g., a news consumer) can check the provenance and authenticity of the content by verifying the associated signatures.

The trust model of C2PA assumes that both recording devices and editing software are trustworthy. However, the trust assumption regarding editing software is problematic in practice: while recording devices may utilize trusted execution environments (TEEs) or hardware security module (HSMs) to protect the signing keys, these mechanisms are not available for editing software. Consequently, attackers could potentially extract signing keys via reverse engineering, enabling them to generate valid signatures for malicious content.

Furthermore, C2PA may inadvertently leak sensitive information. For instance, during the editing process, the thumbnail of the original content might be signed by the editing software and published along with the processed content for verification. This may expose data that was not intended for disclosure, such as the faces of individuals that were blurred in the processed content, thereby raising privacy concerns.

Detection of fake videos. Another direction to fight misinformation is detecting fake videos. Human eyes are not always reliable in distinguishing real videos from fake ones, especially with the rise of deepfake technology [81, 54, 38]. Focusing on the detection of AI-generated videos, machine learning models have been developed [62, 65, 43], achieving promising results. However, the inherent characteristics of human eyes and neural networks inevitably produce false positives and false negatives with non-negligible probability. This is especially evident when active attackers manipulate videos to exploit some vulnerabilities in a specific detection method. For instance, it is demonstrated in [70, 46] that several existing deepfake detectors can be bypassed by adversarial examples.

Appendix B Full Construction of Loua

Now we provide a concrete example by applying our paradigms to Nova [60] and its implementation in the `sonobe` library [75]. The resulting scheme, Loua, supports LogUp and achieves linear time prover, all while maintaining the constant proof size and verifier time in `sonobe`.

B.1 LouaFS

Overview of Nova. In Nova, we consider *committed relaxed R1CS*, which is a variant of the Rank-1 Constraint System (R1CS) [34]. Similar to R1CS, a committed relaxed R1CS over $\mathbb{F}$ with n constraints and m variables (among which 1 variable is constant and l variables are public inputs) is defined by three matrices $\mathsf{CS} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \in (\mathbb{F}^{n\times m}, \mathbb{F}^{n\times m}, \mathbb{F}^{n\times m})$. A witness $\mathbb{W}$ to CS not only consists of the witness $\boldsymbol{w} \in \mathbb{F}^{m-l-1}$ to the original R1CS relation, but also includes an *error term* $\boldsymbol{e} \in \mathbb{F}^n$. The instance $\mathbb{U}$ corresponding to $\mathbb{W}$ is a tuple $(u, \boldsymbol{x}, \overline{W}, \overline{E})$, where $u \in \mathbb{F}$ is a scalar for absorbing constant terms, $\boldsymbol{x} \in \mathbb{F}^l$ is the public input, and $\overline{W}, \overline{E}$ are the commitments to $\boldsymbol{w}, \boldsymbol{e}$ respectively. We say $(\mathbb{U}, \mathbb{W})$ satisfies CS if $\mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{w}, \overline{W}) = 1$, $\mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{e}, \overline{E}) = 1$, and $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = u \cdot \boldsymbol{C}\boldsymbol{v} + \boldsymbol{e}$, where $\boldsymbol{v} = (u, \boldsymbol{x}, \boldsymbol{w})$.

To fold $(\mathbb{U}_1, \mathbb{W}_1)$ and $(\mathbb{U}_2, \mathbb{W}_2)$, the Nova prover first computes the cross term $\boldsymbol{t}$ and sends the commitment $\overline{T}$ to the verifier, who samples and sends back a challenge r. Then, both parties output the folded instance $\mathbb{U}$ by computing the random linear combinations of all components (i.e., $u, \boldsymbol{x}, \overline{W}, \overline{E}$) in $\mathbb{U}_1$ and $\mathbb{U}_2$, with r as the randomness. The prover further output the folded witness $\mathbb{W}$, whose components $\boldsymbol{w}$ and $\boldsymbol{e}$ are also random linear combinations of their counterparts in $\mathbb{W}_1$ and $\mathbb{W}_2$. One can apply the Fiat-Shamir transform [31] to obtain the non-interactive construction.

Algorithm 17: LouaFS

```
Fn LouaFS.$\mathcal{G}(1^\lambda)$:
    return $\mathsf{pp} \leftarrow \mathsf{CM}.\mathcal{K}(1^\lambda)$
Fn LouaFS.$\mathcal{K}(\mathsf{pp}, \mathsf{CS})$:
    return $\mathsf{pk} \coloneqq (\mathsf{pp}, \mathsf{CS}), \mathsf{vk} \coloneqq \perp$
Fn LouaFS.$\mathcal{P}(\mathsf{pk}, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2))$:
    Parse $(\mathsf{ck}, (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})) \coloneqq \mathsf{pk}$
    for $i \in \{1, 2\}$ do
        Parse $(u_i, \boldsymbol{x}_i, \overline{Q}_i, \overline{W}_i, \overline{E}_i) \coloneqq \mathbb{U}_i, (\boldsymbol{q}_i, \boldsymbol{w}_i, \boldsymbol{e}_i) \coloneqq \mathbb{W}_i$
        $\boldsymbol{v}_i \coloneqq (u_i, \boldsymbol{x}_i, \boldsymbol{q}_i, \boldsymbol{w}_i)$
    $\boldsymbol{t} \coloneqq \boldsymbol{A}\boldsymbol{v}_1 \circ \boldsymbol{B}\boldsymbol{v}_2 + \boldsymbol{A}\boldsymbol{v}_2 \circ \boldsymbol{B}\boldsymbol{v}_1 - u_1 \cdot \boldsymbol{C}\boldsymbol{v}_2 - u_2 \cdot \boldsymbol{C}\boldsymbol{v}_1$
    $\overline{T} \leftarrow \mathsf{CM}.\mathcal{C}(\mathsf{ck}, \boldsymbol{t})$
    $r \coloneqq \rho(\mathbb{U}_1, \mathbb{U}_2, \overline{T})$                ▷ Compute challenge
    Fold instances:
        $u \coloneqq u_1 + ru_2, \boldsymbol{x} \coloneqq \boldsymbol{x}_1 + r\boldsymbol{x}_2$
        $\overline{Q} \coloneqq \overline{Q}_1 + r\overline{Q}_2, \overline{W} \coloneqq \overline{W}_1 + r\overline{W}_2, \overline{E} \coloneqq \overline{E}_1 + r\overline{T} + r^2\overline{E}_2$
    Fold witnesses:
        $\boldsymbol{q} \coloneqq \boldsymbol{q}_1 + r\boldsymbol{q}_2, \boldsymbol{w} \coloneqq \boldsymbol{w}_1 + r\boldsymbol{w}_2, \boldsymbol{e} \coloneqq \boldsymbol{e}_1 + r\boldsymbol{t} + r^2\boldsymbol{e}_2$
    return $\mathbb{U} \coloneqq (u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}), \mathbb{W} \coloneqq (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}), \overline{T}$
Fn LouaFS.$\mathcal{V}(\mathsf{vk}, \mathbb{U}_1, \mathbb{U}_2, \overline{T})$:
    for $i \in \{1, 2\}$ do
        Parse $(u_i, \boldsymbol{x}_i, \overline{Q}_i, \overline{W}_i, \overline{E}_i) \coloneqq \mathbb{U}_i$
    $r \coloneqq \rho(\mathbb{U}_1, \mathbb{U}_2, \overline{T})$                ▷ Compute challenge
    Fold instances:
        $u \coloneqq u_1 + ru_2, \boldsymbol{x} \coloneqq \boldsymbol{x}_1 + r\boldsymbol{x}_2$
        $\overline{Q} \coloneqq \overline{Q}_1 + r\overline{Q}_2, \overline{W} \coloneqq \overline{W}_1 + r\overline{W}_2, \overline{E} \coloneqq \overline{E}_1 + r\overline{T} + r^2\overline{E}_2$
    return $\mathbb{U} \coloneqq (u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E})$
```

Construction of LouaFS. Recall that in our first paradigm, the lookup-friendly folding scheme NIFS$^{\mathsf{LU}}$ (Algorithm 1) separates queries (and multiplicities) $\boldsymbol{q}$ from witnesses $\boldsymbol{w}$. Consequently, LouaFS extends Nova by redefining the folding witness as $\mathbb{W} = (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e})$, where $\boldsymbol{q} = ((\alpha_i)_{i=0}^{\mu-1}, (o_j)_{j=0}^{\nu-1}) \in \mathbb{F}^{\mu+\nu}$, $\boldsymbol{w} \in \mathbb{F}^{m-\mu-\nu-l-1}$, and $\boldsymbol{e} \in \mathbb{F}^n$. The corresponding instance becomes $\mathbb{U} = (u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E})$, where the additional term $\overline{Q}$ is the commitment to $\boldsymbol{q}$. When checking satisfiability of CS, the vector of variables in the constraint system is now $\boldsymbol{v} = (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$. For notational convenience, we write the part of field elements in $\mathbb{U}$ as $\mathbb{U}^{\mathbb{F}} = (u, \boldsymbol{x})$, and the part of group elements (commitments) as $\mathbb{U}^{\mathbb{G}} = (\overline{Q}, \overline{W}, \overline{E})$. With the adapted $\mathbb{W}$ and $\mathbb{U}$, LouaFS additionally computes the random linear combinations of $\boldsymbol{q}$ in folding witnesses and $\overline{Q}$ in folding instances, and the full construction is given in Algorithm 17.

Security of LouaFS. We provide the intuition to prove the security of LouaFS in terms of completeness, knowledge soundness, and zero-knowledge.

Essentially, LouaFS modifies Nova by splitting the vector of witnesses into $\boldsymbol{q}$ and $\boldsymbol{w}$. Thus, the core step in our proof is the conversion from the instance-witness pair $\mathbb{U}, \mathbb{W}$ in LouaFS to the one $\mathbb{U}', \mathbb{W}'$ in Nova, where $\mathbb{U}' \coloneqq (\mathbb{U}.u, \mathbb{U}.\boldsymbol{x}, \mathbb{U}.\overline{Q} + \mathbb{U}.\overline{W}, \mathbb{U}.\overline{E})$, $\mathbb{W}' \coloneqq (\mathbb{W}.\boldsymbol{q} \cup \mathbb{W}.\boldsymbol{w}, \mathbb{W}.\boldsymbol{e})$.

Proof of completeness. Given an adversary $\mathcal{A}$ who breaks the completeness of LouaFS, we can construct an adversary $\mathcal{A}'$ who breaks the completeness of Nova. Whenever $\mathcal{A}$ outputs $R, (\mathbb{U}_1, \mathbb{W}_1), (\mathbb{U}_2, \mathbb{W}_2)$, $\mathcal{A}'$ constructs the corresponding instances and witnesses $(\mathbb{U}'_1, \mathbb{W}'_1), (\mathbb{U}'_2, \mathbb{W}'_2)$ for Nova. Then, $\mathcal{A}'$ outputs $R, (\mathbb{U}'_1, \mathbb{W}'_1), (\mathbb{U}'_2, \mathbb{W}'_2)$, thereby breaking the completeness of Nova. $\square$

Proof of knowledge soundness. With Nova's extractor Ext, we build an extractor Ext' for LouaFS. Provided $R, \mathbb{U}_1, \mathbb{U}_2, \mathbb{W}, \overline{T}$, which are the output of $\mathcal{A}$, Ext' converts $\mathbb{U}_1, \mathbb{U}_2, \mathbb{W}$ to Nova's instances and witness $\mathbb{U}'_1, \mathbb{U}'_2, \mathbb{W}'$, feeds $R, \mathbb{U}'_1, \mathbb{U}'_2, \mathbb{W}', \overline{T}$ to Ext, converts the returned $\mathbb{W}'_1, \mathbb{W}'_2$ back to $\mathbb{W}_1, \mathbb{W}_2$, and outputs $\mathbb{W}_1, \mathbb{W}_2$. $\square$

Proof of zero-knowledge. Intuitively, LouaFS is zero-knowledge because the commitment $\overline{T}$ in the transcript is hiding. Formally, the simulator Sim uniformly samples a random value r_t and computes $\overline{T} \leftarrow \mathsf{CM}.\mathcal{C}(\mathsf{ck}, r_t)$. $\overline{T}$ is indistinguishable from honestly generated commitments, because CM is a hiding commitment scheme. $\square$

B.2 LouaIVC

Construction of LouaIVC. The IVC scheme $\mathsf{LouaIVC}$ is then constructed by plugging Nova into $\mathsf{IVC}^{\mathsf{LU}}$ (Algorithm 3). To further reduce the cost of recursive prover, CycleFold [56] is utilized in $\mathsf{LouaIVC}$.

Observe that if we naively instantiate $\mathsf{NIFS}^{\mathsf{LU}}.\mathcal{V}$ as $\mathsf{LouaFS}.\mathcal{V}$ in the augmented circuit $\widetilde{\mathcal{F}}$ (Algorithm 2), then we need to compute in-circuit the random linear combination of commitments, *i.e.*, $\overline{X} := \overline{X}_1 + r\overline{X}_2$, where $\overline{X}$'s coordinates are over the *base field* of $\mathbb{G}$. On the other hand, the circuit is defined over the *scalar field* of $\mathbb{G}$, meaning that expensive non-native operations are necessary to compute $\overline{X}$ in-circuit. A common solution [72] is to deploy two augmented step circuits on a cycle of curves $(\mathbb{G}, \mathbb{H})$, where the circuit on one curve is responsible for folding instances from the other curve. Nevertheless, this approach is suboptimal, as it requires additional costs for encoding the folding verification algorithm on the secondary curve $\mathbb{H}$ as well.

CycleFold [56] aims to minimize the costs on $\mathbb{H}$ by offloading the heavy lifting of non-native operations on the primary curve $\mathbb{G}$ to a lightweight circuit on $\mathbb{H}$, which can handle them natively, thereby avoiding the need for duplicating the entire folding verification algorithm.

With CycleFold, our $\mathsf{LouaFS}.\mathcal{V}$ is split into two parts:

- $\mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}$ folds field elements in $\mathbb{U}_1^{\mathbb{F}}$ and $\mathbb{U}_2^{\mathbb{F}}$.

  On input vk, $\mathbb{U}_1^{\mathbb{F}} = (u_1, \boldsymbol{x}_1)$, $\mathbb{U}_2^{\mathbb{F}} = (u_2, \boldsymbol{x}_2)$ and r, it computes $u := u_1 + ru_2$, $\boldsymbol{x} := \boldsymbol{x}_1 + r\boldsymbol{x}_2$ and returns $\mathbb{U}^{\mathbb{F}} := (u, \boldsymbol{x})$.

- $\mathsf{LouaFS}.\mathcal{V}^{\mathbb{G}}$ folds group elements in $\mathbb{U}_1^{\mathbb{G}}$ and $\mathbb{U}_2^{\mathbb{G}}$.

  On input vk, $\mathbb{U}_1^{\mathbb{G}} = (\overline{Q}_1, \overline{W}_1, \overline{E}_1)$, $\mathbb{U}_2^{\mathbb{G}} = (\overline{Q}_2, \overline{W}_2, \overline{E}_2)$, r and $\overline{T}$, it computes $\overline{Q} := \overline{Q}_1 + r\overline{Q}_2, \overline{W} := \overline{W}_1 + r\overline{W}_2, \overline{E} := \overline{E}_1 + r\overline{T} + r^2\overline{E}_2$ and returns $\mathbb{U}^{\mathbb{G}} := (\overline{Q}, \overline{W}, \overline{E})$.

Then, we construct a CycleFold circuit $\mathcal{F}^{\mathsf{cf}}$ on $\mathbb{H}$ (see Circuit 18) that performs the check $\mathsf{LouaFS}.\mathcal{V}^{\mathbb{G}}$ natively, which requires only ~ 4500 constraints[7]. Denote

[7]In `sonobe`, $\mathcal{F}^{\mathsf{cf}}$ is split into three sub-circuits, each with ~ 1500 constraints, and this is also the case for our variant. We omit this detail for clarify.

$(\mathbb{U}^{\mathsf{cf}}, \mathbb{W}^{\mathsf{cf}})$ and $(\mathbb{u}^{\mathsf{cf}}, \mathbb{w}^{\mathsf{cf}})$ respectively as the running and incoming instance-witness pairs for $\mathcal{F}^{\mathsf{cf}}$.

```
Circuit 18: $\mathcal{F}^{\mathsf{cf}}(r, \mathbb{U}_i^{\mathbb{G}}, \mathbb{u}_i^{\mathbb{G}}, \overline{T}) \to \mathbb{U}_{i+1}^{\mathbb{G}}$

   Statement: $r, \mathbb{U}_i^{\mathbb{G}}, \mathbb{u}_i^{\mathbb{G}}, \mathbb{U}_{i+1}^{\mathbb{G}}, \overline{T}$
 1 return $\mathbb{U}_{i+1}^{\mathbb{G}} \coloneqq \mathsf{LouaFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}, \mathbb{U}_i^{\mathbb{G}}, \mathbb{u}_i^{\mathbb{G}}, r, \overline{T})$
```

```
Circuit 19: $\widetilde{\mathcal{F}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \mathbb{U}_{i+1}^{\mathbb{G}}, \overline{T}, \mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}, \overline{T}^{\mathsf{cf}}, \mathsf{aux}_i) \to (h_1, h_2, c)$

   Witness: $i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbb{u}_i, \mathbb{U}_{i+1}^{\mathbb{G}}, \overline{T}, \mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}, \overline{T}^{\mathsf{cf}}, \mathsf{aux}_i$
   Statement: $h_1, h_2, c$
   Constant: $(\tau_j)_{j=0}^{\nu-1}$
 1 $\boldsymbol{z}_{i+1} \coloneqq \mathcal{F}(\boldsymbol{z}_i; \mathsf{aux}_i)$        ▷ Let $(\alpha_i)_{i=0}^{\mu-1}$ be queries made by $\mathcal{F}$
 2 Check $\mathbb{u}_i$:
       enforce $\mathbb{u}_i.u = 1$
       enforce $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \varrho(\mathbb{U}_i^{\mathsf{cf}}, i), \rho(\mathbb{u}_i.\overline{Q}))$
       enforce $\mathbb{u}_i.\overline{E} = \overline{0}$
 3 $r \coloneqq \rho(\mathbb{U}_i, \mathbb{u}_i, \overline{T})$
 4 $\mathbb{U}_{i+1}^{\mathbb{F}} \coloneqq \mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}, \mathbb{U}_i^{\mathbb{F}}, \mathbb{u}_i^{\mathbb{F}}, r)$
 5 Check $\mathbb{u}_i^{\mathsf{cf}}$:
       enforce $\mathbb{u}_i^{\mathsf{cf}}.u = 1$
       enforce $\mathbb{u}_i^{\mathsf{cf}}.\boldsymbol{x} = (r, \mathbb{U}_i^{\mathbb{G}}, \mathbb{u}_i^{\mathbb{G}}, \mathbb{U}_{i+1}^{\mathbb{G}}, \overline{T})$
       enforce $\mathbb{u}_i^{\mathsf{cf}}.\overline{E} = \overline{0}$
 6 $\mathbb{U}_{i+1}^{\mathsf{cf}} \coloneqq \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}^{\mathsf{cf}}, \mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}, \overline{T}^{\mathsf{cf}})$
 7 Check lookup queries:
       $\boldsymbol{o} \leftarrow \mathsf{Hint}(\boldsymbol{\alpha})$
       $c \leftarrow \mathsf{Hint}(\boldsymbol{\alpha}, \boldsymbol{o})$
       enforce $\sum_{i=0}^{\mu-1} \frac{1}{c-\alpha_i} = \sum_{j=0}^{\nu-1} \frac{o_j}{c-\tau_j}$
 8 Compute public outputs:
       $h_1 \coloneqq \varrho((i=0)\ ?\ \mathbb{U}_\perp : \mathbb{U}_{i+1}, i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1})$
       $h_2 \coloneqq \varrho((i=0)\ ?\ \mathbb{U}_\perp^{\mathsf{cf}} : \mathbb{U}_{i+1}^{\mathsf{cf}}, i+1)$
 9 return $h_1, h_2, c$
```

With the help of $\mathcal{F}^{\mathsf{cf}}$, the augmented circuit $\widetilde{\mathcal{F}}$ only needs to fold the field parts of primary instances. As a trade-off, $\widetilde{\mathcal{F}}$ 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 $\mathsf{LouaFS}.\mathcal{V}$ can be handled natively by $\widetilde{\mathcal{F}}$ over $\mathbb{G}$. Although the *field* elements in $\mathbb{U}_i^{\mathsf{cf}}, \mathbb{u}_i^{\mathsf{cf}}$ become non-native, emulating non-native field operations is much cheaper than non-native group operations, thanks to the techniques in [55]. To summarize, we present the details of LoualVC's $\widetilde{\mathcal{F}}$ in Circuit 19.

Having constructed $\widetilde{\mathcal{F}}$, we illustrate LoualVC in Algorithm 20. In the setup algorithm $\mathsf{LoualVC}.\mathcal{G}$, two commitment keys ck and $\mathsf{ck}^{\mathsf{cf}}$ are generated, one for primary instances, and the other for CycleFold instances. The key generation algorithm $\mathsf{LoualVC}.\mathcal{K}$ takes a step function $\mathcal{F}$ as input, creates the augmented function $\widetilde{\mathcal{F}}$ for $\mathcal{F}$, and converts $\widetilde{\mathcal{F}}$ and $\mathcal{F}^{\mathsf{cf}}$ to R1CS matrices $\widetilde{\mathsf{CS}}$ and $\mathsf{CS}^{\mathsf{cf}}$. Then, $\mathsf{LouaFS}.\mathcal{K}$ is invoked for both matrices to obtain the proving and verification keys.

Before the proof generation $\mathsf{LoualVC}.\mathcal{P}$ starts, the prover first prepares two empty running instance-witness pairs $(\mathbb{U}_0 \coloneqq \mathbb{U}_\perp, \mathbb{W}_0 \coloneqq \mathbb{W}_\perp)$ and $(\mathbb{U}_0^{\mathsf{cf}} \coloneqq \mathbb{U}_\perp^{\mathsf{cf}},$

Algorithm 20: LouaIVC

```
1  Fn LouaIVC.$\mathcal{G}(1^\lambda)$:
2      return $(\mathsf{ck} \leftarrow \mathsf{LouaFS}.\mathcal{G}(1^\lambda), \mathsf{ck}^{\mathsf{cf}} \leftarrow \mathsf{LouaFS}.\mathcal{G}(1^\lambda))$
3  Fn LouaIVC.$\mathcal{K}((\mathsf{ck}, \mathsf{ck}^{\mathsf{cf}}), \mathcal{F})$:
4      Wrap $\mathcal{F}$ and build $\widetilde{\mathcal{F}}$ and $\mathcal{F}^{\mathsf{cf}}$
5      Encode $\widetilde{\mathcal{F}}$ and $\mathcal{F}^{\mathsf{cf}}$ as $\widetilde{\mathsf{CS}}$ and $\mathsf{CS}^{\mathsf{cf}}$ in R1CS
6      $(\mathsf{pk}, \mathsf{vk}) := \mathsf{LouaFS}.\mathcal{K}(\mathsf{ck}, \widetilde{\mathsf{CS}})$
7      $(\mathsf{pk}^{\mathsf{cf}}, \mathsf{vk}^{\mathsf{cf}}) := \mathsf{LouaFS}.\mathcal{K}(\mathsf{ck}^{\mathsf{cf}}, \mathsf{CS}^{\mathsf{cf}})$
8      return $(\mathsf{pk}, \mathsf{pk}^{\mathsf{cf}}), (\mathsf{vk}, \mathsf{vk}^{\mathsf{cf}})$
9  Fn LouaIVC.$\mathcal{P}((\mathsf{pk}, \mathsf{pk}^{\mathsf{cf}}), (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \mathsf{aux}_i, \pi_i)$:
10     Parse $((\mathbb{U}_i, \mathbb{W}_i), (\mathbbm{u}_i, \mathbbm{w}_i), (\mathbb{U}_i^{\mathsf{cf}}, \mathbb{W}_i^{\mathsf{cf}})) := \pi_i$
11     $(\mathbb{U}_{i+1}, \mathbb{W}_{i+1}, \overline{T}) := (i = 0)\ ?\ (\mathbb{U}_\perp, \mathbb{W}_\perp, \overline{0}) : \mathsf{LouaFS}.\mathcal{P}(\mathsf{pk}, (\mathbb{U}_i, \mathbb{W}_i), (\mathbbm{u}_i, \mathbbm{w}_i))$
12     $r := \rho(\mathbb{U}_i, \mathbbm{u}_i, \overline{T})$
13     $\mathbb{U}_{i+1}^{\mathbb{G}} \leftarrow \mathcal{F}^{\mathsf{cf}}(r, \mathbb{U}_i^{\mathbb{G}}, \mathbbm{u}_i^{\mathbb{G}}, \overline{T})$
14     Construct $\mathbbm{u}_i^{\mathsf{cf}}, \mathbbm{w}_i^{\mathsf{cf}}$:
           Extract variables $\boldsymbol{v}^{\mathsf{cf}} = (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$ in $\mathcal{F}^{\mathsf{cf}}$, where $u = 1, \boldsymbol{q} = \varnothing, \boldsymbol{x} = (r, \mathbb{U}_i^{\mathbb{G}}, \mathbbm{u}_i^{\mathbb{G}}, \mathbb{U}_{i+1}^{\mathbb{G}})$
           $\mathbbm{w}_i^{\mathsf{cf}} := (\boldsymbol{q}, \boldsymbol{w}, \varnothing)$,    $\mathbbm{u}_i^{\mathsf{cf}} \leftarrow (u, \boldsymbol{x}, \overline{0}, \mathsf{CM}.\mathcal{C}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{w}), \overline{0})$
15     $(\mathbb{U}_{i+1}^{\mathsf{cf}}, \mathbb{W}_{i+1}^{\mathsf{cf}}, \overline{T}^{\mathsf{cf}}) := (i = 0)\ ?\ (\mathbb{U}_\perp^{\mathsf{cf}}, \mathbb{W}_\perp^{\mathsf{cf}}, \overline{0}) : \mathsf{LouaFS}.\mathcal{P}(\mathsf{pk}^{\mathsf{cf}}, (\mathbb{U}_i^{\mathsf{cf}}, \mathbb{W}_i^{\mathsf{cf}}), (\mathbbm{u}_i^{\mathsf{cf}}, \mathbbm{w}_i^{\mathsf{cf}}))$
16     $(h_1, h_2, c) \leftarrow \widetilde{\mathcal{F}}(i, \boldsymbol{z}_i, \mathbb{U}_i, \mathbbm{u}_i, \mathbb{U}_{i+1}^{\mathbb{G}}, \overline{T}, \mathbb{U}_i^{\mathsf{cf}}, \mathbbm{u}_i^{\mathsf{cf}}, \overline{T}^{\mathsf{cf}}, \mathsf{aux}_i)$
17     Construct $\mathbbm{u}_{i+1}, \mathbbm{w}_{i+1}$:
           Extract variables $\boldsymbol{v} = (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$ in $\widetilde{\mathcal{F}}$, where $u = 1, \boldsymbol{x} = (h_1, h_2, c)$
           $\mathbbm{w}_{i+1} := (\boldsymbol{q}, \boldsymbol{w}, \varnothing)$,    $\mathbbm{u}_{i+1} \leftarrow (u, \boldsymbol{x}, \mathsf{CM}.\mathcal{C}(\mathsf{ck}, \boldsymbol{q}), \mathsf{CM}.\mathcal{C}(\mathsf{ck}, \boldsymbol{w}), \overline{0})$
18     return $\pi_{i+1} := ((\mathbb{U}_{i+1}, \mathbb{W}_{i+1}), (\mathbbm{u}_{i+1}, \mathbbm{w}_{i+1}), (\mathbb{U}_{i+1}^{\mathsf{cf}}, \mathbb{W}_{i+1}^{\mathsf{cf}}))$
19 Fn LouaIVC.$\mathcal{V}((\mathsf{vk}, \mathsf{vk}^{\mathsf{cf}}), (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i)$:
20     Parse $((\mathbb{U}_i, \mathbb{W}_i), (\mathbbm{u}_i, \mathbbm{w}_i), (\mathbb{U}_i^{\mathsf{cf}}, \mathbb{W}_i^{\mathsf{cf}})) := \pi_i$
21     Check $\mathbbm{u}_i$:
           assert $\mathbbm{u}_i.u = 1 \wedge \mathbbm{u}_i.\overline{E} = \overline{0}$
           assert $\mathbbm{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \varrho(\mathbb{U}_i^{\mathsf{cf}}, i), \rho(\mathbbm{u}_i.\overline{Q}))$
22     Check $\mathbbm{w}_i$ against $\mathbbm{u}_i$:
           Parse $(u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}) := \mathbbm{u}_i, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbbm{w}_i$
           $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
           assert $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = \boldsymbol{C}\boldsymbol{v}$
           assert $\mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{q}, \overline{Q}) \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{w}, \overline{W}) \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{e}, \overline{E})$
23     Check $\mathbb{W}_i$ against $\mathbb{U}_i$:
           Parse $(u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}) := \mathbb{U}_i, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}_i$
           $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
           assert $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = u \cdot \boldsymbol{C}\boldsymbol{v} + \boldsymbol{e}$
           assert $\mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{q}, \overline{Q}) \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{w}, \overline{W}) \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}, \boldsymbol{e}, \overline{E})$
24     Check $\mathbb{W}_i^{\mathsf{cf}}$ against $\mathbb{U}_i^{\mathsf{cf}}$:
           Parse $(u, \boldsymbol{x}, \overline{Q}, \overline{W}, \overline{E}) := \mathbb{U}_i^{\mathsf{cf}}, (\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{e}) := \mathbb{W}_i^{\mathsf{cf}}$
           $\boldsymbol{v} := (u, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{w})$
           assert $\boldsymbol{A}^{\mathsf{cf}}\boldsymbol{v} \circ \boldsymbol{B}^{\mathsf{cf}}\boldsymbol{v} = u \cdot \boldsymbol{C}^{\mathsf{cf}}\boldsymbol{v} + \boldsymbol{e}$
           assert $\boldsymbol{q} = \varnothing \wedge \overline{Q} = \overline{0} \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{w}, \overline{W}) \wedge \mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{e}, \overline{E})$
25     return 1
```

$\mathbb{W}_0^{\mathsf{cf}} := \mathbb{W}_\perp^{\mathsf{cf}}$). The incoming instance-witness pair in the 0-th step is also ($\mathbbm{u}_0 := \mathbb{U}_\perp, \mathbbm{w}_0 := \mathbb{W}_\perp$). The prover and verifier then proceed to the incremental proof generation and verification, in the same way as the generic ones in Section 4.1, but with additional logic to handle CycleFold circuits and instance-witness pairs.

Security of $\mathsf{LoualVC}$. Below we provide proofs of the security of $\mathsf{LoualVC}$.

Proof of succinctness. $\mathsf{LoualVC}$ is succinct, because π_i only consists of two running instance-witness pairs and one incoming instance-witness pair, whose sizes are independent of the number of steps. □

Proof of completeness. Now we prove the completeness of $\mathsf{LoualVC}$. We focus on the non-base case where $i > 0$ but omit the base case $i = 0$, where the proof is trivial.

Recall that $\mathsf{LoualVC}$ is knowledge sound, if for $(\mathcal{F}, i, \boldsymbol{z}_0, \boldsymbol{z}_i, \mathsf{aux}_i, \pi_i)$ output by any efficient adversary $\mathcal{A}$ that satisfies $\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1$, the prover can always output a proof π_{i+1} that satisfies $\mathcal{V}(\mathsf{vk}, (i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1}), \pi_{i+1}) = 1$.

By the condition $\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1$ and the construction of $\mathsf{LoualVC}.\mathcal{V}$, we learn that:

- $\mathbb{u}_i$ is a valid incoming instance with $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \varrho(\mathbb{U}_i^{\mathsf{cf}}, i), \rho(\mathbb{u}_i.\overline{Q}))$,
- $\mathbb{w}_i$ and $\mathbb{u}_i$ satisfy $\widetilde{\mathsf{CS}}$,
- $\mathbb{W}_i$ and $\mathbb{U}_i$ satisfy $\widetilde{\mathsf{CS}}$, and
- $\mathbb{W}_i^{\mathsf{cf}}$ and $\mathbb{U}_i^{\mathsf{cf}}$ satisfy $\mathsf{CS}^{\mathsf{cf}}$

Our goal is to show that $\mathcal{V}(\mathsf{vk}, (i+1, \boldsymbol{z}_0, \boldsymbol{z}_{i+1}), \pi_{i+1}) = 1$. According to the construction of $\mathsf{LoualVC}.\mathcal{P}$, $\mathbb{U}_{i+1}$ and $\mathbb{W}_{i+1}$ are obtained by folding $\mathbb{u}_i, \mathbb{w}_i$ into $\mathbb{U}_i, \mathbb{W}_i$. Since LouaFS is complete, $\mathbb{U}_{i+1}$ and $\mathbb{W}_{i+1}$ also satisfy $\widetilde{\mathsf{CS}}$.

Moreover, when running $\mathcal{F}^{\mathsf{cf}}$ in $\mathsf{LoualVC}.\mathcal{P}$, the check in Circuit 18 passes due to the completeness of LouaFS. Therefore, $\mathbb{u}_i^{\mathsf{cf}}$ and $\mathbb{w}_i^{\mathsf{cf}}$, constructed from the variables in $\mathcal{F}^{\mathsf{cf}}$, satisfy $\mathsf{CS}^{\mathsf{cf}}$. As a result, both $(\mathbb{u}_i^{\mathsf{cf}}, \mathbb{w}_i^{\mathsf{cf}})$ and $(\mathbb{U}_i^{\mathsf{cf}}, \mathbb{W}_i^{\mathsf{cf}})$ satisfy $\mathsf{CS}^{\mathsf{cf}}$. Again, by the completeness of LouaFS, $\mathbb{U}_{i+1}^{\mathsf{cf}}$ and $\mathbb{W}_{i+1}^{\mathsf{cf}}$ also satisfy $\mathsf{CS}^{\mathsf{cf}}$.

To complete the proof, we demonstrate that $\mathbb{u}_{i+1}$ and $\mathbb{w}_{i+1}$ satisfy $\widetilde{\mathsf{CS}}$ as well, where $\mathbb{u}_{i+1}$ is a valid incoming instance. Consider the checks in Circuit 19. First, Line 2 passes because $\mathbb{u}_i$ is known to be valid. In addition, according to the construction of $\mathbb{u}_i^{\mathsf{cf}}$, Line 6 is also satisfied. Line 7 checks the equation for set inclusion, which holds since the queries $\boldsymbol{\alpha}$ are supposed to be a subset of the lookup table $\boldsymbol{\tau}$. Moreover, Line 8 ensures that the statements h_1 and h_2 match the in-circuit variables calculated via $\mathsf{LouaFS}.\mathcal{V}$ and ϱ, which is guaranteed by the completeness of LouaFS. Consequently, the vector of variables $\boldsymbol{z}$ is valid for $\widetilde{\mathcal{F}}$, implying that $\mathbb{u}_{i+1}$ and $\mathbb{w}_{i+1}$ satisfy $\widetilde{\mathsf{CS}}$. Since c is computed as $c := \rho(\mathbb{u}_{i+1}.\overline{Q})$, it follows that $\mathbb{u}_{i+1}.\boldsymbol{x} = (h_1, h_2, c)$ is well-formed. □

Proof of knowledge soundness. Below we show that $\mathsf{LoualVC}$ satisfies knowledge soundness. Again, we only consider the non-base case with $i > 1$.

Recall that $\mathsf{LoualVC}$ is knowledge sound, if for $(\mathcal{F}, (i \geq 1, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i)$ output by any efficient adversary $\mathcal{A}$ that satisfies $\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1$, there is an efficient extractor Ext who can output $(\boldsymbol{z}_{i-1}, \mathsf{aux}_{i-1}, \pi_{i-1})$ that satisfies the two checks below:

- $\boldsymbol{z}_i = \mathcal{F}(\boldsymbol{z}_{i-1}; \mathsf{aux}_{i-1})$

- $\mathcal{V}(\mathsf{vk}, (i-1, \boldsymbol{z}_0, \boldsymbol{z}_{i-1}), \pi_{i-1}) = 1$

By the condition $\mathcal{V}(\mathsf{vk}, (i, \boldsymbol{z}_0, \boldsymbol{z}_i), \pi_i) = 1$ and the construction of $\mathsf{LouaIVC}.\mathcal{V}$, we learn that:

- $\mathbb{u}_i$ is a valid incoming instance with $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \varrho(\mathbb{U}^{\mathsf{cf}}_i, i), \rho(\mathbb{u}_i.\overline{Q}))$,
- $\mathbb{w}_i$ and $\mathbb{u}_i$ satisfy $\widetilde{\mathsf{CS}}$,
- $\mathbb{W}_i$ and $\mathbb{U}_i$ satisfy $\widetilde{\mathsf{CS}}$, and
- $\mathbb{W}^{\mathsf{cf}}_i$ and $\mathbb{U}^{\mathsf{cf}}_i$ satisfy $\mathsf{CS}^{\mathsf{cf}}$

Consequently, we can construct an Ext who works as below:

1. Reconstruct $\boldsymbol{v}$ from $\mathbb{w}_i$ and $\mathbb{u}_i$ by computing $\boldsymbol{v} := (\mathbb{u}_i.u, \mathbb{u}_i.\boldsymbol{x}, \mathbb{w}_i.\boldsymbol{q}, \mathbb{w}_i.\boldsymbol{w})$.

   Because $\mathbb{w}_i$ and $\mathbb{u}_i$ satisfy $\widetilde{\mathsf{CS}}$, $\boldsymbol{v}$ is a satisfying vector of variables for $\widetilde{\mathcal{F}}$.

2. Obtain the witnesses $j, \boldsymbol{z}_j, \mathbb{U}_j, \mathbb{u}_j, \mathbb{U}^{\mathbb{G}}_{j+1}, \overline{T}_j, \mathbb{U}^{\mathsf{cf}}_j, \mathbb{u}^{\mathsf{cf}}_j, \overline{T}^{\mathsf{cf}}_j$ from $\boldsymbol{v}$.

   - By the checks in Line 8 of Circuit 19, we have $h_1 = \varrho(\mathbb{U}_{j+1}, j+1, \boldsymbol{z}_0, \boldsymbol{z}_{j+1})$, and $h_2 = \varrho(\mathbb{U}^{\mathsf{cf}}_{j+1}, j+1)$. Also, since h_1, h_2 are parts of $\mathbb{u}_i.\boldsymbol{x} = (\varrho(\mathbb{U}_i, i, \boldsymbol{z}_0, \boldsymbol{z}_i), \varrho(\mathbb{U}^{\mathsf{cf}}_i, i), \rho(\mathbb{u}_i.\overline{Q}))$ and ϱ is collision-resistant, we can deduce that the two preimages are equal, except with negligible probability, *i.e.*,
     - $j + 1 = i$.
     - $\mathbb{U}^{\mathsf{cf}}_{j+1} = \mathbb{U}^{\mathsf{cf}}_i$.
       Combining with Line 6 of Circuit 19, we have $\mathbb{U}^{\mathsf{cf}}_i = \mathbb{U}^{\mathsf{cf}}_{j+1} = \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}^{\mathsf{cf}}, \mathbb{U}^{\mathsf{cf}}_j, \mathbb{u}^{\mathsf{cf}}_j, \overline{T}^{\mathsf{cf}}_j)$.
     - $\mathbb{U}_{j+1} = \mathbb{U}_i$.
     - $\boldsymbol{z}_{j+1} = \boldsymbol{z}_i$.
       Combining with Line 1 of Circuit 19, we have $\boldsymbol{z}_i = \boldsymbol{z}_{j+1} = \mathcal{F}(\boldsymbol{z}_j; \mathsf{aux}_j)$.
   - By the checks in Line 2 of Circuit 19, we know that $\mathbb{u}_j$ is an incoming instance, and that $\mathbb{u}_j.\boldsymbol{x} = (\varrho(\mathbb{U}_j, j, \boldsymbol{z}_0, \boldsymbol{z}_j), \varrho(\mathbb{U}^{\mathsf{cf}}_j, j), \rho(\mathbb{u}_j.\overline{Q}))$.
   - By the checks in Line 7 of Circuit 19, LogUp's identity for set inclusion holds, given a uniform challenge $c := \rho(\mathbb{u}_i.\overline{Q})$. According to [42, Lemma 5], the queries are hence in the lookup table, *i.e.*, $\boldsymbol{\alpha} \subseteq \boldsymbol{\tau}$.

3. Invoke the extractor of LouaFS with input $\mathsf{CS}^{\mathsf{cf}}, \mathbb{U}^{\mathsf{cf}}_j, \mathbb{u}^{\mathsf{cf}}_j, \mathbb{W}^{\mathsf{cf}}_i, \overline{T}^{\mathsf{cf}}_j$.

   Due to the knowledge soundness of LouaFS as well as the facts that $\mathbb{U}^{\mathsf{cf}}_i = \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}^{\mathsf{cf}}, \mathbb{U}^{\mathsf{cf}}_j, \mathbb{u}^{\mathsf{cf}}_j, \overline{T}^{\mathsf{cf}}_j)$ and $(\mathbb{U}^{\mathsf{cf}}_i, \mathbb{W}^{\mathsf{cf}}_i)$ satisfy $\mathsf{CS}^{\mathsf{cf}}$, LouaFS's extractor is able to return $\mathbb{W}^{\mathsf{cf}}_j$ and $\mathbb{w}^{\mathsf{cf}}_j$ such that both $(\mathbb{U}^{\mathsf{cf}}_j, \mathbb{W}^{\mathsf{cf}}_j)$ and $(\mathbb{u}^{\mathsf{cf}}_j, \mathbb{w}^{\mathsf{cf}}_j)$ satisfy $\mathsf{CS}^{\mathsf{cf}}$, except with negligible probability.

4. Reconstruct $\boldsymbol{v}^{\mathsf{cf}}$ from $\mathbb{w}^{\mathsf{cf}}_i$ and $\mathbb{u}^{\mathsf{cf}}_i$ by computing $\boldsymbol{v}^{\mathsf{cf}} := (\mathbb{u}^{\mathsf{cf}}_i.u, \mathbb{u}^{\mathsf{cf}}_i.\boldsymbol{x}, \mathbb{w}^{\mathsf{cf}}_i.\boldsymbol{q}, \mathbb{w}_i.\boldsymbol{w})$.

   Because $\mathbb{w}^{\mathsf{cf}}_i$ and $\mathbb{u}^{\mathsf{cf}}_i$ satisfy $\mathsf{CS}^{\mathsf{cf}}$, $\boldsymbol{v}^{\mathsf{cf}}$ is a satisfying vector of variables for $\mathcal{F}^{\mathsf{cf}}$.

Due to the checks in Line 5 of Circuit 19, the statement in $\boldsymbol{v}^{\mathsf{cf}}$ is $\mathbb{u}_j^{\mathsf{cf}}.\boldsymbol{x} = (r_j, \mathbb{U}_j^{\mathbb{G}}, \mathbb{u}_j^{\mathbb{G}}, \mathbb{U}_{j+1}^{\mathbb{G}}, \overline{T}_j)$. Combining this with the check in Circuit 18, we can conclude that $\mathbb{U}_{j+1}^{\mathbb{G}} = \mathsf{NIFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}, \mathbb{U}_j^{\mathbb{G}}, \mathbb{u}_j^{\mathbb{G}}, r_j, \overline{T}_j)$. Also, Lines 3-4 of Circuit 19 ensure that $\mathbb{U}_{j+1}^{\mathbb{F}} = \mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}, \mathbb{U}_j^{\mathbb{F}}, \mathbb{u}_j^{\mathbb{F}}, r_j)$ and $r_j = \rho(\mathbb{U}_j, \mathbb{u}_j, \overline{T}_j)$.

Thus, $\mathbb{U}_{j+1} = \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_j, \mathbb{u}_j, \overline{T}_j)$. Note that we already have $\mathbb{U}_i = \mathbb{U}_{j+1}$, we can deduce that $\mathbb{U}_i = \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_j, \mathbb{u}_j, \overline{T}_j)$.

5. Invoke the extractor of LouaFS with input $\widetilde{\mathsf{CS}}, \mathbb{U}_j, \mathbb{u}_j, \mathbb{W}_i, \overline{T}_j$.

   Due to the knowledge soundness of LouaFS as well as the facts that $\mathbb{U}_i = \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}, \mathbb{U}_j, \mathbb{u}_j, \overline{T}_j)$ and $(\mathbb{U}_i, \mathbb{W}_i)$ satisfy $\widetilde{\mathsf{CS}}$, LouaFS's extractor is able to return $\mathbb{W}_j$ and $\mathbb{w}_j$ such that both $(\mathbb{U}_j, \mathbb{W}_j)$ and $(\mathbb{u}_j, \mathbb{w}_j)$ satisfy $\widetilde{\mathsf{CS}}$, except with negligible probability.

6. Compute $\pi_j := ((\mathbb{U}_j, \mathbb{W}_j), (\mathbb{u}_j, \mathbb{w}_j), (\mathbb{U}_j^{\mathsf{cf}}, \mathbb{W}_j^{\mathsf{cf}}))$ and output $\boldsymbol{z}_j, \mathsf{aux}_j, \pi_j$ as well as $\boldsymbol{\alpha}$.

We can observe from the analysis above that the outputs of Ext satisfy the checks in IVC's knowledge soundness definition and the lookup relation R^{lookup}, thereby concluding the proof.

$\square$

B.3 LouaDecider

Construction of LouaDecider. Finally, we introduce the decider LouaDecider and its associated circuit $\mathcal{F}^{\mathsf{Decider}}$ for Loua. Recall that the paradigm in Section 4.2 requires the decider circuit to run the verification algorithm of IVC, but without verifying the commitments in-circuit. In our case, $\mathcal{F}^{\mathsf{Decider}}$ needs to check the primary instance-witness pair $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1})$ and the CycleFold instance-witness pair $(\mathbb{U}_k^{\mathsf{cf}}, \mathbb{W}_k^{\mathsf{cf}})$.

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 $\mathbb{G}$, while 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. This is not covered in Section 4.2, which focuses on eliminating non-native group operations. However, we observe that 1) non-native field operations are relatively cheap in comparison to non-native group operations, and 2) the size of $\mathcal{F}^{\mathsf{cf}}$ (and thus $\mathsf{CS}^{\mathsf{cf}}$) is constant and small. Thus, checking the $\mathsf{CS}^{\mathsf{cf}}$ satisfiability in-circuit is still feasible, and we further apply several optimizations to improve the efficiency, whose details are presented in Section 6.

We summarize the construction of the decider circuit $\mathcal{F}^{\mathsf{Decider}}$ in Circuit 21, and the corresponding decider LouaDecider in Algorithm 22.

Security of LouaDecider. We show that LouaDecider is a zkSNARK for R^{IVC}. For completeness, knowledge soundness, and zero-knowledge, we refer the reader to [60, Appendix D]. Due to the similar design, these security properties can be demonstrated in the same way as Nova's decider. In a nutshell, we can use straightforward reductions to show that if $\mathcal{A}$ is able to break these properties, then the corresponding properties of LegoGro16 and LouaFS would also be broken. Plus, LouaDecider is succinct because both the LegoGro16 proof ϖ and the Pedersen commitments in committed instances are of constant size.

Circuit 21: $\mathcal{F}^{\mathsf{Decider}}(\mathbb{W}_{k+1}, \mathbb{W}^{\mathsf{cf}}_k, \mathbb{U}^{\mathbb{F}}_{k+1}, \mathbb{U}^{\mathsf{cf}}_k)$

```
   Witness: $\mathbb{W}_{k+1}, \mathbb{W}^{\mathsf{cf}}_k$
   Statement: $\mathbb{U}^{\mathbb{F}}_{k+1}, \mathbb{U}^{\mathsf{cf}}_k$
   Constant: $\widetilde{\mathsf{CS}} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}), \mathsf{CS}^{\mathsf{cf}} = (\boldsymbol{A}^{\mathsf{cf}}, \boldsymbol{B}^{\mathsf{cf}}, \boldsymbol{C}^{\mathsf{cf}}), \mathsf{ck}^{\mathsf{cf}}$
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 $\boldsymbol{A}\boldsymbol{v} \circ \boldsymbol{B}\boldsymbol{v} = u \cdot \boldsymbol{C}\boldsymbol{v} + \boldsymbol{e}$
2  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 $\boldsymbol{q} = \varnothing \land \overline{Q} = \overline{0}$
       enforce $\mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{w}, \overline{W})$
       enforce $\mathsf{CM}.\mathcal{V}(\mathsf{ck}^{\mathsf{cf}}, \boldsymbol{e}, \overline{E})$
```

Algorithm 22: LouaDecider

```
1  Fn $\mathsf{LouaDecider}.\mathcal{K}(1^\lambda, (\mathsf{ck}, \mathsf{CS}^{\mathsf{LouaDecider}}))$:
2      return $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{LegoGro16}.\mathcal{K}(1^\lambda, \mathsf{ck}, \mathsf{CS}^{\mathsf{LouaDecider}})$
3  Fn $\mathsf{LouaDecider}.\mathcal{P}((\mathsf{pk}, \mathsf{pk}_\Phi), (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k)$:
4      Parse $((\mathbb{U}_k, \mathbb{W}_k), (\mathbbm{u}_k, \mathbbm{w}_k), (\mathbb{U}^{\mathsf{cf}}_k, \mathbb{W}^{\mathsf{cf}}_k)) := \pi_k$
5      $(\mathbb{U}_{k+1}, \mathbb{W}_{k+1}, \overline{T}) := \mathsf{LouaFS}.\mathcal{P}(\mathsf{pk}_\Phi, (\mathbb{U}_k, \mathbb{W}_k), (\mathbbm{u}_k, \mathbbm{w}_k))$
6      $\boldsymbol{x} := (\mathbb{U}^{\mathbb{F}}_{k+1}, \mathbb{U}^{\mathsf{cf}}_k), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$
7      $\boldsymbol{v} := (\mathbb{W}_{k+1}), \boldsymbol{\omega} := (\mathbb{W}^{\mathsf{cf}}_k)$
8      $\varpi \leftarrow \mathsf{LegoGro16}.\mathcal{P}(\mathsf{pk}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{v}, \boldsymbol{\omega})$
9      return $(\varpi, \mathbb{U}_k, \mathbbm{u}_k, \mathbb{U}^{\mathsf{cf}}_k, \overline{T})$
10 Fn $\mathsf{LouaDecider}.\mathcal{V}((\mathsf{vk}, \mathsf{vk}_\Phi), (k, \boldsymbol{z}_0, \boldsymbol{z}_k), (\varpi, \mathbb{U}_k, \mathbbm{u}_k, \mathbb{U}^{\mathsf{cf}}_k, \overline{T}))$:
11     $\mathbb{U}_{k+1} := \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_k, \mathbbm{u}_k, \overline{T})$
12     Check $\mathbbm{u}_k$:
           assert $\mathbbm{u}_k.u = 1, \mathbbm{u}_k.\overline{E} = \overline{0}$
           assert $\mathbbm{u}_k.\boldsymbol{x} = (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, \boldsymbol{z}_k), \varrho(\mathbb{U}^{\mathsf{cf}}_k, k), \rho(\mathbbm{u}_k.\overline{Q}))$
13     $\boldsymbol{x} := (\mathbb{U}^{\mathbb{F}}_{k+1}, \mathbb{U}^{\mathsf{cf}}_k), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$
14     return $\mathsf{LegoGro16}.\mathcal{V}(\mathsf{vk}, \boldsymbol{x}, \boldsymbol{c}, \varpi)$
```

Appendix C Gadgets for Integer Operations

We review the gadgets in [28] for the computation of sign and absolute value as well as the right shifting operation.

Sign and absolute value. We cannot directly compute the sign and the absolute value of a variable x in an arithmetic circuit over $\mathbb{F}_p$. Intuitively, a number is positive if it is greater than 0, and is negative otherwise. However, as the field $\mathbb{F}_p$ is not *ordered*, we cannot *compare* between its elements. Therefore, we manually define elements in the set $\{1, 2, \ldots, (p-1)/2\}$ as positive, and those in the set $\{(p+1)/2, \ldots, p-2, p-1\}$ as negative.

Before explaining the computation of sign and absolute value under this definition, we introduce $\mathcal{F}^{\mathsf{EnforceBitLen}}$, a gadget for ensuring the bit length of a variable x is at most W, i.e., $x \in [0, 2^W - 1]$. Powered by lookup arguments, $\mathcal{F}^{\mathsf{EnforceBitLen}}$ is the key to efficiency of the in-circuit operations in quantization.

As depicted in Gadget 23, on input a variable x and a constant W, the gadget first asks the prover to provide $(x_i)_{i=0}^{W/\log\nu-1}$, the limbs of x in base-ν (recall that ν is the size of the lookup table). Here, we assume ν is a power of 2. Then, the gadget enforces x indeed decomposes into these limbs by comparing their concatenation, expressed as $\sum_{i=0}^{W/\log\nu-1} 2^{i\log\nu} x_i$, with x. Finally, the limbs of x are appended to $\boldsymbol{\alpha}$, the list of queries, to make sure every limb is in base-ν. We reemphasize that lookup argument is critical to the performance of $\mathcal{F}^{\mathsf{EnforceBitLen}}$: without the lookup table, circuits in R1CS can only handle the range check bit-by-bit (instead of limb-by-limb), which is done by enforcing $x_i(1-x_i)=0$ for each claimed bit x_i. Consequently, $\mathcal{F}^{\mathsf{EnforceBitLen}}$ would cost $W+1$ constraints, which is much more expensive than the $W/\log\nu+1$ constraints with the lookup table.

Gadget 23: $\mathcal{F}^{\mathsf{EnforceBitLen}}(x, W)$

```
1 $(x_i)_{i=0}^{W/\log\nu-1} \leftarrow \mathsf{Hint}(x)$
2 **enforce** $\sum_{i=0}^{W/\log\nu-1} 2^{i\log\nu} x_i = x$
3 $\boldsymbol{\alpha} := (\boldsymbol{\alpha}, (x_i)_{i=0}^{W/\log\nu-1})$
```

Now, as long as the upper bound of x's absolute value satisfies $x < 2^W < (p-1)/2$, we can extract the sign and the absolute value of x using $\mathcal{F}^{\mathsf{SignAbs}}$, as depicted in Gadget 24. The gadget first asks the prover to determine if x is positive. The prover checks which set x belongs to, and provides s as a hint. The gadget enforces that s is boolean, and computes x's absolute value $y := s\,?\,x : -x$. Finally, the gadget enforces that y has at most W bits by invoking $\mathcal{F}^{\mathsf{EnforceBitLen}}(y, W)$ and returns s and y. Soundness holds because if an adversary feeds the incorrect s to the gadget, then y's value belongs to the negative set and is hence greater than $(p-1)/2$, but $\mathcal{F}^{\mathsf{EnforceBitLen}}$ guarantees that $0 \le y < 2^W < (p-1)/2$.

Gadget 24: $\mathcal{F}^{\mathsf{SignAbs}}(x \in [-2^W+1, 2^W-1])$

```
1 $s \leftarrow \mathsf{Hint}(x)$
2 $y := s\,?\,x : -x$
3 **enforce** $s(1-s) = 0$
4 $\mathcal{F}^{\mathsf{EnforceBitLen}}(y, W)$
5 **return** $s, y$
```

Right shift. It is also non-trivial to implement the gadget $\mathcal{F}^{\gg}(x, \delta)$ for shifting x to the right by δ bits. Here, we assume that $x \in [0, 2^W-1]$, $\delta \in [U, V]$, and $2^{W+V-U} < p$. Intuitively, we could treat the right shift operation as integer division, i.e., $x \gg \delta = x/2^\delta$. The prover computes the quotient q and the remainder r such that $x = q \cdot 2^\delta + r$, and feeds q, r as hints to the gadget. Then the gadget enforces $x = q \cdot 2^\delta + r$. In addition, it is also required to check that $q \in [0, 2^{W-U}-1], r \in [0, 2^\delta - 1]$ to ensure q and r are well-formed. Here, since δ is not a constant, it requires two $\mathcal{F}^{\mathsf{EnforceBitLen}}$ calls to enforce r's range, one for checking $r \in [0, 2^V-1]$ and another for checking $2^\delta - 1 - r \in [0, 2^V-1]$, introducing $2V/\log\nu$ queries to the lookup table. We are convinced that $r \in [0, 2^\delta-1]$ only when both conditions are satisfied.

However, it is possible to eliminate one $\mathcal{F}^{\mathsf{EnforceBitLen}}$ call. As presented in Gadget 25, $\mathcal{F}^{\gg}$ first computes $x' := x \ll (V-\delta) = x \cdot 2^{V-\delta}$. Since $\delta \in [U, V]$, we have $V - \delta \in [0, V-U]$, and thus $x \cdot 2^{V-\delta} < 2^{W+V-U} < (p-1)/2$ does not overflow. Then we handle $x' \gg V$ analogously: the prover provides the quotient q

and the remainder r for $x'/2^V$ as hints, and the gadget checks if $q \in [0, 2^{W-U}-1]$, $r \in [0, 2^V - 1]$, and $x' = q \cdot 2^V + r$. This optimized approach only adds $V/\log \nu$ queries to the lookup table for checking r, thereby saving $V/\log \nu$ constraints compared to the naive construction.

Gadget 25: $\mathcal{F}^{\gg}(x \in [0, 2^W - 1], \delta \in [U, V])$

```
1 $x' := x \cdot 2^{V-\delta}$
2 $q, r \leftarrow \mathsf{Hint}(x')$
3 **enforce** $x' = q \cdot 2^V + r$
4 $\mathcal{F}^{\mathsf{EnforceBitLen}}(q, W - U)$
5 $\mathcal{F}^{\mathsf{EnforceBitLen}}(r, V)$
```

Appendix D 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 LegoGro16 proof ϖ has 4 $\mathbb{G}$ elements and 1 $\widehat{\mathbb{G}}$ element, the partial running instance $\mathbb{U}'_k$ has 3 $\mathbb{G}$ elements and 1 $\mathbb{F}_p$ element, the partial incoming instance $\mathbb{u}'_k$ 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 $\widehat{\mathbb{G}}$ 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 LouaFS, LouaIVC, 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 $\boldsymbol{x} := (\mathsf{vk}_\Sigma, \mathsf{meta}, k, \boldsymbol{z}_0, \hbar_k, r, \mathbb{u}'_k, \mathbb{U}'_k, \overline{T}), \boldsymbol{c} := (\mathbb{U}^{\mathbb{G}}_{k+1})$. With this condition, Ext works as below:

1. Invoke the extractor of ZKCP 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 $\mathbb{u}_k$ from $\mathbb{u}'_k$ and $\mathbb{u}_k.\boldsymbol{x} := (\varrho(\mathbb{U}_k, k, \boldsymbol{z}_0, (h_k, \hbar_k)), \varrho(\mathbb{U}^{\mathsf{cf}}_k, k), \rho(\mathbb{u}_k.\overline{Q}))$.

4. Lines 3-4 of Circuit 15 enforce that $\mathbb{U}^{\mathbb{F}}_{k+1} := \mathsf{LouaFS}.\mathcal{V}^{\mathbb{F}}(\mathsf{vk}_\Phi, \mathbb{U}^{\mathbb{F}}_k, \mathbb{u}^{\mathbb{F}}_k, r)$ and $r = \rho(\mathbb{U}_k, \mathbb{u}_k, \overline{T})$. Also, we have $\mathbb{U}^{\mathbb{G}}_{k+1} := \mathsf{LouaFS}.\mathcal{V}^{\mathbb{G}}(\mathsf{vk}_\Phi, \mathbb{U}^{\mathbb{G}}_k, \mathbb{u}^{\mathbb{G}}_k, r, \overline{T})$, due to the construction of Eva's verifier $\mathcal{V}$. Thus, $\mathbb{U}_{k+1} := \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_k, \mathbb{u}_k, \overline{T})$.

   Moreover, Line 5 of Circuit 15 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 $\widetilde{\mathsf{CS}}^{\mathsf{Eva}}$.

   Consequently, except with negligible probability, Ext can invoke the extractor of LouaFS on input $\mathbb{U}_k, \mathbb{u}_k, \mathbb{W}_{k+1}, \overline{T}$ and obtain $\mathbb{W}_k, \mathbb{w}_k$ such that both $(\mathbb{U}_k, \mathbb{W}_k)$ and $(\mathbb{u}_k, \mathbb{w}_k)$ satisfy $\widetilde{\mathsf{CS}}^{\mathsf{Eva}}$.

5. By the checks in Line 6 of Circuit 15, we can deduce that $\mathbb{U}_k^{\mathsf{cf}}$ and $\mathbb{W}_k^{\mathsf{cf}}$ satisfy $\mathsf{CS}^{\mathsf{cf}}$. At this point, Ext can recover $\pi_k := ((\mathbb{U}_k, \mathbb{W}_k), (\mathbb{u}_k, \mathbb{w}_k), (\mathbb{U}_k^{\mathsf{cf}}, \mathbb{W}_k^{\mathsf{cf}}))$ such that all checks in $\mathsf{LouaIVC}.\mathcal{V}(\mathsf{vk}_\Phi, (k, \boldsymbol{z}_0, \boldsymbol{z}_k), \pi_k) = 1$ pass.

   Consequently, except with negligible probability, Ext can invoke the extractor of $\mathsf{LouaIVC}$ with input $\widetilde{\mathcal{F}}^{\mathsf{Eva}}, k, \boldsymbol{z}_0, \boldsymbol{z}_k, \pi_k$ and obtain the state and proof at $k-1$-th step.

6. Repeatedly invoke the extractor of $\mathsf{LouaIVC}$ 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 $\widetilde{\mathcal{F}}^{\mathsf{Eva}}$, 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 15, 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, thereby completing the proof. $\square$

Proof of zero-knowledge. For zero-knowledge, we leverage the technique in [60, Appendix D] to construct a simulator Sim who can produce $\mathbb{U}_k, \mathbb{u}_k$ that are indistinguishable from the outputs of the honest prover, if ϱ and CM are hiding.

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

Next, we show that for every i, if $\mathbb{U}_i$ and $\mathbb{u}_i$ are indistinguishable from real ones, then Sim can generate $\mathbb{U}_{i+1}$ and $\mathbb{u}_{i+1}$ 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 $\mathbb{U}_{i+1} := \mathsf{LouaFS}.\mathcal{V}(\mathsf{vk}_\Phi, \mathbb{U}_i, \mathbb{u}_i, \overline{T})$. Further, $\mathbb{u}_{i+1}$ is derived in the same way as the base case. In this way, both $\mathbb{U}_{i+1}$ and $\mathbb{u}_{i+1}$ are indistinguishable from real instances.

After k steps, $\mathbb{U}_k, \mathbb{u}_k$ 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 $r := \rho(\mathbb{U}_k, \mathbb{u}_k, \overline{T})$, which are indistinguishable from real ones.

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

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

Discussion. For a raw video $\boldsymbol{V}$ signed by the recorder and an encoded video stream ζ, our current security model guarantees that $\zeta = \mathcal{E}(\Delta(\boldsymbol{V}), \mathsf{param}^{\mathcal{E}})$, where Δ is the editing operation, and $\mathsf{param}^{\mathcal{E}}$ is the encoding parameters. Below we discuss two potential issues with our current security model and possible solutions.

First, our model does not ensure $\mathsf{param}^{\mathsf{Pred}}$ to be the best prediction parameters for encoding $\boldsymbol{V}' := \Delta(\boldsymbol{V})$. Recall that, in order to reduce the circuit size, the prediction process is removed from our $\mathcal{F}^{\mathcal{E}}$, and thus the choice of prediction parameters $\mathsf{param}^{\mathsf{Pred}}$ is not enforced.

Consequently, for an original macroblock $\boldsymbol{X} \in \boldsymbol{V}$ and two editing operations Δ and $\widehat{\Delta}$, a malicious prover $\mathcal{A}$ may produce $\boldsymbol{P}$ and $\boldsymbol{Z}$ by encoding $\boldsymbol{X}' := \Delta(\boldsymbol{X})$, but later prove that $\boldsymbol{Z}$ is encoded from $\widehat{\boldsymbol{X}}' := \widehat{\Delta}(\boldsymbol{X})$. This is possible if $\mathcal{A}$ can find some prediction parameters $\widehat{\mathsf{param}}^{\mathsf{Pred}}$ such that the prediction macroblock for $\widehat{\boldsymbol{X}}'$ becomes $\widehat{\boldsymbol{P}} = \widehat{\boldsymbol{X}}' - \boldsymbol{X}' + \boldsymbol{P}$. By feeding $\widehat{\boldsymbol{P}}$ to the circuit, the residual macroblock is now computed as $\widehat{\boldsymbol{X}}' - \widehat{\boldsymbol{P}} = \boldsymbol{X}' - \boldsymbol{P} = \boldsymbol{R}$, and the final output becomes $\boldsymbol{Z}$. In this way, $\mathcal{A}$'s claimed editing operation is $\widetilde{\Delta} = \widehat{\Delta}$, and the claimed encoding parameters are $\widetilde{\mathsf{param}}^{\mathsf{Pred}} = \widehat{\mathsf{param}}^{\mathsf{Pred}}$, but $\boldsymbol{Z}$ is actually the encoding of $\Delta(\boldsymbol{X})$ under $\mathsf{param}^{\mathsf{Pred}}$.

Our model allows for such an "attack", because both $\Delta, \mathsf{param}^{\mathsf{Pred}}$ and $\widehat{\Delta}$, $\widehat{\mathsf{param}}^{\mathsf{Pred}}$ are valid configurations for encoding $\boldsymbol{X}$ as $\boldsymbol{Z}$. A stronger security model might require the claimed editing operation $\widetilde{\Delta}$ to closely resemble the actual one Δ (as we will discuss soon, it is impossible to guarantee exactly equality between $\widetilde{\Delta}$ and Δ).

To achieve security under this enhanced model, we propose the following approaches:

- If the video codec generates similar predictions for different prediction parameters, or if we can restrict the prover to use prediction parameters with similar effects, then $\mathcal{A}$ can no longer find a prediction macroblock that balances the large difference between the claimed $\widetilde{\Delta}$ and the actual Δ. This would ensure that $\widetilde{\Delta} \approx \Delta$.

- If the above conditions are not met, it is still possible to mitigate this issue. Observe that while lossy encoding reduces the video quality, the encoded video ζ is still similar to the original $\boldsymbol{V}'$. Thus, with the correct prediction parameters $\widetilde{\mathsf{param}}^{\mathsf{Pred}} = \mathsf{param}^{\mathsf{Pred}}$ for $\boldsymbol{V}'$, encoding ζ under $\mathsf{param}^{\mathsf{Pred}}$ again will produce a video that resembles ζ. On the other hand, with cheating prediction parameters $\widetilde{\mathsf{param}}^{\mathsf{Pred}} \not\approx \mathsf{param}^{\mathsf{Pred}}$, the re-encoded video will significantly differ from ζ.

  Therefore, to detect if $\mathcal{A}$ is cheating, the verifier can re-encode ζ with the claimed prediction parameters and inspect if the re-encoded video is close to the encoded video. By rejecting significantly different videos, $\mathcal{V}$ can ensure that the claimed prediction parameters satisfy $\widetilde{\mathsf{param}}^{\mathsf{Pred}} \approx \mathsf{param}^{\mathsf{Pred}}$, thereby guaranteeing $\widetilde{\Delta} \approx \Delta$.

- The most robust solution is to extend Eva and generate proofs of best encoding. Intuitively, one can achieve this by incorporating the prediction process into the circuit. However, designing a more efficient approach remains an open problem.

The second issue is that the quantization parameter qp may enable a malicious prover $\mathcal{A}$ to find two editing operations Δ and $\widetilde{\Delta}$ that produce the same video stream ζ after quantization.

We regard the design of a stronger security model that guarantees the strict equality between Δ and $\widetilde{\Delta}$ as out of scope. The reason is that, due to the inherent loss of information in lossy encoding, it is always possible for $\mathcal{A}$ to find two modifications on the original video that have the same encoded stream, even qp is fixed to some small values. In some scenarios such as the detection of fake news, the verifier can simply reject low-quality videos with large qp (e.g., $\mathsf{qp} \geq 50$), as we expect videos published by news agencies to have more reasonable qp values (e.g., $\mathsf{qp} \approx 30$).