Trustless Delegation of Vector Commitment Construction in Resource-Constrained Settings

Parisa Hassanizadeh[1,2], Shahriar Ebrahimi[2,3], Stefan Dziembowski[4,5] and Janusz Szczepanski[1]

[1] IPPT Polish Academy of Science (PAN), Poland, jszczepa@ippt.pan.pl
phassani@ippt.pan.pl
[2] Zero Savvy, United Kingdom, parisa@zerosavvy.xyz, shahriar@zerosavvy.xyz
[3] The Alan Turing Institute, United Kingdom, sebrahimi@turing.ac.uk
[4] University of Warsaw, Poland, stefan.dziembowski@crypto.edu.pl
[5] IDEAS Institute, Warsaw, Poland

Abstract. Many data types, such as video and audio, consist of sequential elements where both integrity and order are essential for authenticity. In practice, verifiers often access only partial sequences due to privacy or bandwidth constraints, motivating the use of vector commitments (VCs) for verifiable partial disclosure. However, VC construction requires a trusted committer, typically within a secure module on the device, and maintaining such commitments is challenging for resource-constrained hardware. For example, in CCTV pipelines, a trusted module processing continuous streams must store and update large VC structures, leading to prohibitive memory and computational overhead.

This work addresses this deployment bottleneck by introducing an efficient pipeline for verifiable VC construction that offloads computation from constrained devices while preserving trust. The source device computes and signs a cumulative hash over the data stream, requiring only constant memory. Later, an untrusted prover reconstructs the VC from the raw data and produces a zero-knowledge proof that the construction is consistent with the signed hash chain. This design eliminates the need for trusted storage of intermediate VC state and enables verifiable partial disclosures from the reconstructed VC.

A key challenge is the high cost of proving the full VC construction. We address this by designing a folding-based zkSNARKs system tailored to streaming workloads. We implement and evaluate the system on a constrained device (Raspberry Pi Zero) as the source and a consumer-grade prover (midrange laptop). Our results show that direct VC maintenance on the source device requires hundreds of megabytes of memory and is computationally infeasible on trusted platform standards for even moderate workloads (e.g., 30 minutes of video). In contrast, our approach reduces the trusted device's memory footprint to constant size, while the midrange laptop can generate the proof of full VC construction in approximately 2 minutes for the same workload. Furthermore, the proof size is around 10 KB regardless of the original size of data and verification time is sub-second. Our implementation is available open source at: https://github.com/zero-savvy/proven-view.

Keywords: zkSNARKs · Provenance · Vector Commitments · Folding Schemes

1 Introduction

The authenticity and accuracy of many modern systems that involve decision-making, logical reasoning, or system behaviour analysis, heavily rely on big data that can be structured sequentially, where integrity depends not only on the content of individual

elements but also on their order. Therefore, preserving this order is essential for various data types, including media files, financial records, sensor outputs, and health monitoring data. Meanwhile, the emergence of generative AI and deepfakes has intensified concerns around misinformation by enabling the low-cost creation of synthetic content. As a result, the authenticity of diverse data types, in particular video and audio recordings, has become an increasing concern in both academia and industry.

Authenticating these sequential data requires a reliable binding to their original source, ideally generated by a trusted entity such as a tamper-proof module in a camera [LYC+24, son22, Qua20]. Existing industry methods primarily rely on digital signatures, which can effectively certify the authenticity of entire sequence [LNS+22]; However, this means that in order to verify, the verifier would always need to have access to the entire source. On the other hand, recordings often contain extra content unrelated to the core event, presenting a significant challenge: "*Authenticating and verifying arbitrary trims without revealing the rest of the original sequence.*" This is particularly critical in scenarios where privacy concerns restrict full disclosure of the source, such as court proceedings, investigative journalism, and whistleblower cases.

Initiatives such as the C2PA[1] propose embedding metadata into media to certify origin and editing history [c2p24b]. This metadata is authenticated via digital signatures issued by trusted components, often implemented within Trusted Execution Environments (TEEs) [c2p24b]. Systems such as Photoshop and DALL-E have adopted this approach [C2P24a]. However, TEEs introduce additional trust assumptions, restrict verification to specific hardware platforms, and remain vulnerable to a range of attacks [CSB+26, JSS20, CLZ23, AB20]. These limitations stem from reliance on complex TEE environments (e.g., SGX), which support large applications but expand the attack surface. In contrast, more constrained hardware such as TPMs[2] offers stronger tamper resistance but cannot support such workloads. This motivates minimizing in-TEE computation to enable deployment on simpler, more secure hardware.

An alternative direction is to use techniques such as zero-knowledge proofs (ZKPs) to certify edits without relying on TEEs, assuming the original data is trusted [DCB24, DEH25, ZYO+24]. These approaches require a binding commitment to the source, typically realised via vector commitments (VCs) such as Merkle trees [TB23, PSG+24], enabling verifiable partial disclosure. However, practical challenges remain. First, archival formats (e.g., CCTV footage) introduce variability in frame sizes and encoding, complicating integrity verification. Second, prior work often overlooks the computational cost of VC construction itself as pointed in studies like [DCB24, LYC+24].

This limitation is especially pronounced in embedded and edge settings. For example, a trusted capture device such as a camera or sensor node is typically equipped with a secure element or a TPM that supports only limited memory and streaming computation. As discussed in Section 2.1 (Table 1), TPM modules can efficiently maintain a cumulative hash over incoming data but cannot store or update large cryptographic data structures such as VCs. Our experiments (refer to Section 7) confirm that even for moderate workloads (e.g., 30 minutes of video), direct VC construction exceeds the available memory and computational budget of such devices.

In this paper, we address this deployment bottleneck by decoupling VC construction from trusted data capture. The source device maintains only a chained hash of the data stream, which can be computed by constrained hardware, while an untrusted prover constructs the VC. To ensure integrity, the prover must generate a proof that the VC is consistent with the authenticated hash chain produced by the trusted source. As a result, the trusted component only needs to compute and sign the final chained hash, enabling deployment on TPM-class devices. The main challenge is that proving correct

[1] Coalition for Content Provenance and Authenticity
[2] Trusted Platform Modules

VC construction over large sequences remains computationally expensive[3], as the proof must capture the full commitment process.

We implement the full protocol as an open-source library and analyze its security under a well-defined system and adversary model. We evaluate it in a realistic setting where the data source is resource-constrained (Raspberry Pi Zero) and the prover runs on a mid-range laptop. The results show that direct VC construction on the source is infeasible due to memory and computation limits. In contrast, our approach reduces the trusted device's memory usage to constant size and enables the prover to generate proofs for moderate workloads (e.g., 30 minutes of video) in approximately two minutes.

The key contributions of this paper are as follows:

- **Trustless delegation of VC construction for constrained devices.** To the best of our knowledge, we present the first system that offloads VC construction from resource-constrained sources while preserving end-to-end verifiability. The source maintains only a low-complexity cumulative hash over the data stream.
- **Efficient ZK proving of VC construction.** To address the high cost of proving VC construction in ZK, we design a folding-based proof system based on Nova [KST22]. Our approach enables practical proof generation on consumer-grade devices without requiring substantial memory or computation, making full VC construction feasible in settings where general-purpose zkSNARKs such as Groth16 [Gro16] are impractical.
- **Authenticity of arbitrary trims.** Our proof system supports efficient verification of trimmed segments. The proof complexity is independent of the trim size and location, and scales logarithmically with the size of the original source ($O(\log s)$).
- **Implementation and evaluation.** We implement the full protocol as an open-source library[4] and evaluate it in a realistic setting, where a mid-range laptop acts as the prover and a *Raspberry Pi* serves as the data source.

The remainder of the paper is organized as follows. Section 2 reviews the background. Section 3 surveys related work, outlines target applications, and describes the system and adversary model. Section 4 presents our VC scheme and its trustless construction based on a chained hash. Section 5 shows how our VC enables efficient authenticity proofs for partial disclosures. Section 6 details the protocol implementation and circuit design. Section 7 provides our experimental results. Finally, Section 8 concludes the paper.

2 Background

2.1 Vector Commitment

VCs are cryptographic primitives that function as an accumulator, enabling a committer to bind to an ordered sequence of values. This allows for selective openings at specific positions, while a verifier can confirm the authenticity of individual elements within the committed vector without the need for having access to the entire sequence [CF13, TB23, BBF19].

Definition 1. A VC scheme can be defined as a quadruple of PPT algorithms $VC = (\mathsf{KeyGen}_{\Delta}, \mathsf{Com}_{\Delta}, \mathsf{Open}_{\Delta}, \mathsf{Vfy}_{\Delta})$ as follows:

[3]For context, constructing a Merkle tree over a 15-minute low-frame-rate video requires proving approximately $56\text{K} = 30 \times 60 \times 30 \times 2$ hash evaluations within a ZK circuit. Even when using zk-friendly hash functions such as Poseidon [GKR+21], this results in over 13 million constraints in compilers like Circom [Gro16, BMIMT+22], which exceeds the practical limits of consumer-level devices due to memory requirements during proof generation. Increasing the frame rate or video length further amplifies this cost, quickly rendering the construction infeasible even on more capable hardware.

[4]https://github.com/zero-savvy/proven-view

- $pp \leftarrow \mathsf{KeyGen}_{\Delta}(1^{\lambda}, n)$: The generator outputs public parameters pp based on the security parameter λ, and message space size of n.
- $\Delta_M \leftarrow \mathsf{Com}_{\Delta}(pp, m_1, \ldots, m_n)$: Commits to all elements in the sequence $\{m_i\}_{i\in[n]}$.
- $\delta_k \leftarrow \mathsf{Open}_{\Delta}(pp, m_k, k, \text{aux})$: Produces a proof for opening the commitment corresponding to the k^{th} element in the sequence, where $k \in [n]$. The opening usually requires knowledge of some auxiliary value aux.
- $b \leftarrow \mathsf{Vfy}_{\Delta}(pp, \Delta_M, m_k, k, \delta_k)$: The verifier accepts the proof δ_k iff m_k is indeed the k^{th} value of the accumulated Δ_M.

VC schemes can also support efficient updatability and batch proofs [BBF19, TB23], however, for simplicity, we focus on the basic definition while noting that these features remain compatible with our constructions. We require a VC to satisfy ***position-binding***, meaning for every index i in the range $[n]$ and for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ successfully opens the commitment at index i with two different values is negligible [CF13].

2.2 Non-Interactive Zero-Knowledge Arguments of Knowledge

We follow the original definition from [Gro16] to formally define Non-Interactive Zero-Knowledge Arguments of Knowledge.

Definition 2. Let $\mathcal{R}$ be a relation generator that, given a security parameter λ in unary, outputs a binary relation R that is decidable in polynomial time. For any pair $(\phi, w) \in R$, we refer to ϕ as the *statement* and w as the *witness*. Let $\mathcal{R}_{\lambda}$ denote the set of relations that $\mathcal{R}$ may output when given input 1^{λ}. Now, we define a publicly verifiable non-interactive argument system for $\mathcal{R}$ as a quadruple of probabilistic polynomial algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prv}, \mathsf{Vfy}, \mathsf{Sim})$ such that:

- $(\sigma, \tau) \leftarrow \mathsf{Setup}(R)$: produces the CRS σ and a trapdoor τ for the relation R.
- $\pi \leftarrow \mathsf{Prv}(R, \sigma, \phi, w)$: given that $(\phi, w) \in R$, the algorithm returns an argument π.
- $b \leftarrow \mathsf{Vfy}(R, \sigma, \phi, \pi)$: using the same statement ϕ and CRS σ used in generating π, the algorithm Vfy returns $b \in \mathbb{Z}_2$: 0 (reject) or 1 (accept).
- $\pi \leftarrow \mathsf{Sim}(R, \tau, \phi)$: Given the trapdoor τ and statement ϕ, Sim returns argument π.

Definition 3. According to [Gro16], we can label Π as a "*perfect non-interactive zero-knowledge argument of knowledge*" if it satisfies following properties: **Perfect Completeness**: An honest prover with valid witness should convince an honest verifier. Formally:

$$\Pr\left[\mathsf{Vfy}(R, \sigma, \phi, \pi) = 1 \;\middle|\; \begin{array}{c} (\sigma, \tau) \leftarrow \mathsf{Setup}(R) \\ (\phi, w) \in R\,, \quad \pi \leftarrow \mathsf{Prv}(R, \sigma, \phi, w) \end{array}\right] = 1$$

Definition 4. Computational Soundness: The Π is sound if it is not possible to prove a false statement. Let L_R denote the language consisting of all statements such as ϕ that have matching witness in R. Therefore, for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{c}\mathsf{Vfy}(R, \sigma, \phi, \pi) = 1, \\ \phi \notin L_R\end{array} \;\middle|\; \begin{array}{c}(R, z) \leftarrow \mathcal{R}(1^{\lambda}), \quad (\sigma, \tau) \leftarrow \mathsf{Setup}(R) \\ (\phi, \pi) \leftarrow \mathcal{A}(R, z, \sigma)\end{array}\right] = \text{negl}(\lambda)$$

Computational Knowledge Soundness: To call Π an "*argument of knowledge*", we require existence of a PPT extractor $\mathcal{X}_{\mathcal{A}}$ for all PPT adversaries $\mathcal{A}$ that can compute the witness whenever the adversary manages to provide a valid argument:

$$\Pr\left[\begin{array}{c}\mathsf{Vfy}(R, \sigma, \phi, \pi) = 1, \\ (\phi, w) \notin R\end{array} \;\middle|\; \begin{array}{c}(R, z) \leftarrow \mathcal{R}(1^{\lambda}), \quad (\sigma, \tau) \leftarrow \mathsf{Setup}(R) \\ ((\phi, \pi), w) \leftarrow (\mathcal{A} \| \mathcal{X}_{\mathcal{A}})(R, z, \sigma)\end{array}\right] = \text{negl}(\lambda)$$

Definition 5. Perfect Zero-knowledge: The argument does not leak anything beyond the truth of the statement. We say Π is perfect zero-knowledge if for all adversaries $\mathcal{A}$ with access to z, which is an auxiliary side information derived from relation generator:

$$\Pr\left[\mathcal{A}(R,z,\sigma,\tau,\pi)=1 \,\middle|\, \begin{matrix}(\sigma,\tau)\leftarrow\mathsf{Setup}(R), \quad (\phi,w)\in R \\ \pi\leftarrow\mathsf{Prv}(R,\sigma,\phi,w)\end{matrix}\right]=\Pr\left[\mathcal{A}(R,z,\sigma,\tau,\pi)=1 \,\middle|\, \begin{matrix}(\sigma,\tau)\leftarrow\mathsf{Setup}(R), \quad (\phi,w)\in R \\ \pi\leftarrow\mathsf{Sim}(R,\tau,\phi)\end{matrix}\right]$$

Incrementally Verifiable Computation (IVC). Folding-based zkSNARKs are a set of specific cryptographic primitives that achieve IVC by allowing verification of computations done by repeated application of the same function. More precisely, for a given function f, with initial input z_0, an IVC scheme generates a proof Π_i for stating that $z_i = f(z_{i-1}) = f^i(z_0)$, given a proof Π_{i-1} for stating $z_{i-1} = f^{i-1}(z_0)$. An interesting property of IVC schemes in general is that they support additional auxiliary inputs for f. While the main input for each iteration of applying f comes from previous step, the auxiliary input ω_i in each step is independent of other steps. Nova [KST22] was one of the first to construct a compressed zkSNARK from an IVC scheme. To achieve this, they introduced the concept of folding schemes, which transform the verification of two NP statements into checking a single NP statement. The result was a final compressed Spartan zkSNARK, which is a transparent SNARK, meaning that it does not require any trusted setup [Set20]. The `Nova-snark` library [nov24a] provides one of the fastest provers for the Nova protocol. This library supports multiple frontends to define steps in Nova, such as Circom [BMIMT+22] through the `nova-scotia` library [nov24b].

3 Related Work and Target Applications

We begin this section with a brief introduction to TPM standards in industry and then provide an overview of the related work. We then introduce the target application scenarios and system assumptions that motivate the design choices of our approach.

3.1 Trustworthy and Tamper-Proof Camera Systems

Modern approaches to trustworthy video capture rely on tamper-proof hardware components such as TPMs [tpm26] to establish root-of-trust. TPMs are primarily deployed in attestation protocols [EH24, HLE+26] rather than in application-level primitives and are designed to perform a limited set of cryptographic operations securely. Their use in constrained environments like embedded cameras introduces fundamental limitations in terms of computation, storage, and throughput. Table 1 provides a detailed comparison of the most widely used TPM chip architectures in industry.

TPMs are equipped with volatile (RAM) and non-volatile (NV) memory. The internal RAM is typically used for transient keys, session data, and temporary buffers during operations like hashing or signing. However, this memory is not directly accessible to users and is severely limited. For example, the number of transient objects that can be loaded simultaneously ranges between 3 and 10, depending on the vendor. NV storage is also highly constrained, typically ranging from 32 to 64 KB. These limits make it impractical to assume such modules could potentially be employed by application-level cryptographic needs in the source device for example to maintain data structures such as VCs.

To mitigate these constraints, TPM software stacks (TSS) implement several techniques such as context swapping, NV storage emulation, and policy digest offloading. For instance, large inputs (e.g., video segments or measurement logs) must be processed in a streaming fashion using sequence-based operations like `TPM2_HashSequenceStart` and `TPM2_SequenceUpdate`. Ultimately, only compact outputs—such as cumulative digests or policy digests—are kept within the TPM. This architectural design aligns with our

Table 1: TPM Storage Limitations

Feature	Infineon SLB 9670	ST ST33T-PHF2ESPI	Nuvoton NPCT750	Intel PTT
NV Storage	64 KB	34 KB	58 KB	Emulated
Max Cmd	4096 B	4096 B	4096 B	4096 B
Max Resp.	5120 B	5120 B	5120 B	5120 B
NV Buf.	2048 B	1024 B	2048 B	1024–2048 B
Trans. Obj.	5	3–5	∼5	N/A
ECC	Yes	Yes	Yes	Yes

assumption that trustworthy camera modules can, at best, compute and sign a chained hash over sequential data, rather than maintaining structures like VCs internally.

Recent academic efforts have also investigated hardware-based secure video pipelines. ProvCam [LYC+24] is a self-contained camera module that is capable of authenticating video footage during capture. ProvCam implements a custom TCB on an FPGA and chains video frames using cumulative hashing. Its design emphasizes minimizing tampering throughout the hardware-software pipeline, achieving greater efficiency in application-specific scenarios compared to TPM-based solutions. These findings reinforce our assumption that trustworthy camera systems are inherently constrained.

3.2 Related Work

Visual Fingerprinting Methods. Both watermarking and perceptual hashing aim to enable post hoc identification of media content by embedding or deriving resilient signatures. Watermarking embeds invisible markers during generation, with recent methods [FXTB22, LTLB24, FCJ+23] demonstrating partial robustness to common transformations. However, their reliability decreases under combined edits such as cropping with compression or contrast adjustment [FEYM24, HHZ+24, MJZ23]. Perceptual hashing, in contrast, computes fingerprints based on visual features, allowing similar outputs for visually similar inputs [SYG+25]. Despite its use in real-world systems such as Apple's CSAM detection [app21], perceptual hashing remains sensitive to adversarial changes and may leak content information [SHNK22].

Digital Signatures. If the point of media capture is assumed to be trusted (e.g., a TEE or TPM or any other trusted hardware module [LYC+24]), digital signature schemes can be effectively used to enable provenance verification of the original raw media by embedding metadata and signing the content [c2p24b, LYC+24]. This approach underpins frameworks such as C2PA [c2p24b], which has been adopted by major industry players including Adobe, Microsoft, and more recently, OpenAI [C2P24a]. However, a key limitation is that signatures apply only to the original media: even simple, non-invasive transformations (such as resizing from 8K to HD) produce outputs without valid signatures. This limitation stems from the requirement that the editor entity is required to be both trusted and tamper-proof. While trusting a hardware module within a camera is already nontrivial, C2PA-based workflows go further by assuming the integrity of the editing environment (e.g., Photoshop running inside a TEE), which introduces additional attack surfaces [VSSY+24]. Consequently, many industry systems [LYC+24, LNS+22, c2p24b] face barriers to widespread adoption, since maintaining a trusted environment is also required during post-processing. This is particularly problematic in domains such as journalism, where edits are not merely aesthetic but often essential for protecting privacy, for example, by redacting individuals at risk.

Zero-Knowledge Proofs. In contrast to C2PA-based approaches, recent research has explored the use of ZKPs to eliminate the need to trust editing software or tools. Systems such as VIMz [DEH25] and VerITAS [DCB24] demonstrate that ZKPs can authenticate

edits in images at resolutions up to 8K. Extending these techniques to video, however, introduces new challenges due to the sequential nature of frames. For example, Zhang et al. [ZYO+24] investigate ZKP-based video authentication for edits such as *redaction* and *blurring* on lossy-encoded videos.

We note that our focus in this paper is orthogonal to prior work, as we target the authenticity of *trims*, a temporal operation typically performed prior to any spatial edits. Specifically, we concentrate on assigning a proper verifiable VC to a sequence that can later be leveraged by methods such as [ZYO+24]. Without a proper VC to handle trims before editing, approaches like [ZYO+24] would exhibit computational complexity that scales with the full length of the source video. In contrast, our construction enables subsequent techniques to operate directly on the trimmed segment, achieving complexity that is independent of the original duration.

VC Schemes. Several studies extend the functionality and scalability of VCs beyond simple Merkle tree structures, either by modifying parent node constructions or by introducing new methods for efficient updatability. Pointproofs [GRWZ20] introduce a VC that supports non-interactive aggregation of proofs across multiple commitments. Reckle [PSG+24] presents a VC construction based on succinct recursive arguments combined with Merkle trees, making it particularly suitable for Map/Reduce-style computations. In [TB23], the authors analyze the trade-off between update information size and proof update time in dynamic VCs, and propose a family of schemes with sublinear update complexity. All of these approaches are compatible with our construction and can be integrated into the framework. However, in our implementation, we use a Merkle tree (§ 6).

3.3 Desired Properties and Real World Constraints

In many applications across domains such as finance, healthcare, and journalism, sensitive information must be selectively disclosed without compromising privacy. While spatial edits such as redaction can protect privacy, temporal trimming is typically performed beforehand, which requires constructing VCs over large media streams. A key challenge is that continuous sources, such as CCTV recordings, generate data that must be incrementally incorporated into the VC. Maintaining and updating such commitments is computationally and storage intensive, particularly under the constraints of real-world devices (§ 3.1).

Moreover, most related work [DEH25, DCB24, ZYO+24] assumes access to a VC construction using ZK-friendly primitives (e.g., Poseidon [GKR+21]), which are not yet widely supported in hardware, unlike standardized primitives such as SHA. Consequently, assuming that commodity trusted devices (e.g., cameras) can construct and maintain fully usable VCs during recording is unrealistic (§ 3.1 and § 7, Table 5). Prior work [LYC+24] similarly observes that even custom secure camera modules struggle to maintain a chained hash at standard video rates, let alone a full VC. This motivates designs that rely only on the ability of trusted devices to sign cumulative hashes of media sequences.

It is worth noting that the C2PA approach shifts trust to editing applications running inside TEEs, thereby introducing a large trusted computing base (TCB). This expands the attack surface and makes the system vulnerable to compromises in both the application layer and the underlying trusted hardware [JSS20, CSB+26].

Overall, a practical solution should satisfy the following requirements:

- **Format agnosticism**: Operate across heterogeneous media formats and compression schemes, including mixed frame types in video streams.
- **Practical deployment**: Support long sequences with minimal computational and storage overhead at the constrained source device (e.g. CCTV camera), while enabling trustless selective disclosure on consumer-grade hardware (e.g. midrange laptop).

- **Efficient partial disclosures**: Enable trustless authentication of partial disclosures (i.e., trims) of the original sequence.

3.4 System and Adversary Model

Figure 1 illustrates the system model and its main actors as follows.

- **Source ($\mathcal{S}$).** A device or entity that produces the original sequence and embeds a verifiable signature in it. This may be an authenticated camera [c2p24b, LYC+24] or a certified content creation tool [C2P24a][5]. We assume that $\mathcal{S}$ securely signs the chained hash of the sequence using a private key corresponding to a publicly known identity [LYC+24]. In practice, $\mathcal{S}$ signs metadata containing the hash of the original sequence and potentially additional information, such as timestamps or location data. Since we do not assume trust in such auxiliary metadata, these fields are outside the scope of our work.
- **Prover ($\mathcal{P}$).** An untrusted party in possession of a sequence, produced by $\mathcal{S}$, who wishes to convince the verifier of the authenticity of certain trims of the sequence, without revealing the original sequence. The prover may be fully adversarial and attempt to manipulate the revealed sequence or its accompanying proof.
- **Verifier ($\mathcal{V}$).** An entity that receives zero-knowledge proofs from $\mathcal{P}$ and checks that (i) the provided signature is authentic, (ii) the provided VC root is indeed derived from the authentic signature, and (iii) the final provided trim belongs to the authenticated VC. Unlike in traditional centralized provenance systems, $\mathcal{V}$ does not query any third-party server; all verification is performed locally given the public keys of trusted origins.

Our setup follows prior work [DCB24, DEH25, MHE25, ZYO+24] and industry standards such as C2PA [c2p24b]. We adopt a standard probabilistic polynomial-time (PPT) adversary model, consistent with prior work [DEH25, DCB24, ZYO+24]. The adversary $\mathcal{A}$ may control the prover $\mathcal{P}$, the communication channel, or both. In particular, $\mathcal{A}$:

- Can observe, intercept, modify, delay, or inject messages between any two parties.
- May attempt to forge VCs or ZKPs or generate a ZKP for an input different from the actual underlying input.
- May try to produce multiple valid but conflicting proofs for the same video trim, or replay previously valid proofs in a different context.
- May adaptively choose inputs, queries, and proofs to maximize the probability of verifier acceptance without possessing a valid authenticated trim.

However, we assume $\mathcal{A}$ cannot:

- Break the unforgeability of the digital signature scheme used by $\mathcal{S}$.
- Find collisions in the cryptographic hash function used in the VC construction.
- Break the soundness or zero-knowledge properties of the underlying zkSNARK construction (e.g., Groth16 or Nova).
- Tamper the trusted module (e.g. TPM chip) in $\mathcal{S}$ to access the private signing key.
- Access original sequence during capture before it reaches the trusted module. This is consistent with both industry standards [c2p24b], and prior work [LYC+24, DCB24].

[5]Examples include Sony Alpha 7 cameras operating in *in-camera signing* mode [son22] and the Snapdragon 888 mobile processor, which provides signing capabilities compatible with C2PA [Qua20]. Alternatively, systems such as [LYC+24] can serve as custom trusted camera modules in this setting.

Figure 1: System model.

Table 2: Terminology of the Paper

Notation	Description
$X = \bigcup x_i$	A sequence of n elements, where for a small $p \in \mathbb{N}$, $\forall i : x_i \in \mathbb{Z}_p^{s_i}$, i.e. $\lvert x_i \rvert = s_i \times \lfloor \log p \rfloor$.
H	A collision-resistant hash, e.g., Poseidon [GKR+21]. $H : \mathbb{Z}_p^2 \to \mathbb{Z}_p$
H_ϕ	Array/frame hasher (§4.1 Definition 6). $H_\phi : \mathbb{Z}_p^s \to \mathbb{Z}_p$
H_Φ	The sequence hasher (§4.1 Definition 6). $H_\Phi([x_0, ..., x_i]) = H(H_\Phi([x_0, ..., x_{i-1}]) \Vert H_\phi(x_i))$.
ℓ_x^i	Cumulative hash of the original sequence until the i^{th} element.
σ_m	Digital signature of message m.
Δ_M	Root of the Merkle tree built over the vector $M = \{m_1, ..., m_n\}$.
δ_i	Merkle proof of the i^{th} leaf.
π	zkSNARK proof: we use notation such as $\pi_\mathcal{T}$, π_σ, and π_Δ.
Vfy	Denotes verification algorithm. We use notations such as Vfy_Δ, Vfy_σ and Vfy_π.

4 Provable Construction of Vector Commitments

We start this section by defining a sequence hashing mechanism based on any collision-resistant block hash. Next, we demonstrate how to build a VC on top of this mechanism to achieve efficient proofs for partial reveals. Finally, we show how to trustlessly prove the construction of the VC using folding-based zkSNARKs. Table 2 provides a summary of our key symbols and terminologies.

4.1 Lightweight Commitments at the Source

We aim to minimize the computational overhead at source device. A realistic assumption is that the device maintains only a cumulative hash, updated as new elements are generated [LYC+24]. At the same time, it must remain possible to commit to and later authenticate individual elements in the sequence, rather than only the dataset as a whole.

To achieve this, we adopt a structure based on *sequential chaining of element hashes*. This provides *position-binding* while allowing commitment to be updated on the fly: each new element is hashed, and the result extends the cumulative hash. A similar approach is also suggested in prior work [LYC+24]. In many scenarios, such as video streams or network logs, element sizes may vary depending on the format. Thus, the hashing method must handle variable-sized inputs and remain compatible with diverse formats.

Definition 6. Let s_i denote the size of i-th element in sequence $X = \{x_1, \ldots, x_n\}$, i.e., $s_i = \lvert x_i \rvert$. A ***format-agnostic sequence hasher*** H_Φ is an algorithm that assigns a cumulative hash value to each $x_i \in X$, incorporating the hash of previous element to maintain sequential integrity. Such a hash function can be constructed using any collision-resistant block hash function H, as illustrated in Figure 2, defined as follows:

$$H_\phi : \mathbb{Z}_p^{s_i} \to \mathbb{Z}_p \;\Big|\; H_\phi(x_i) = H(...H(x_i[1] \Vert x_i[2]) \Vert ... \Vert x_i[s_i])) \to h_x^i$$

$$H_\Phi : \mathbb{Z}_p^{s_i+1} \to \mathbb{Z}_p \;\Big|\; H_\Phi\big(\ell_x^{i-1}, x_i\big) = H\big(\ell_x^{i-1} \Vert h_x^i\big) \to \ell_x^i$$

As illustrated in Figure 2, the cumulative hash of sequence requires $\sum s_i$ evaluations of H. The storage cost is $O(1)$, specifically $(2 \times \log p)$ bits: one variable to store the latest cumulative state ℓ_x^i and one temporary variable to compute $H_\phi(x_i)$ for the current element.

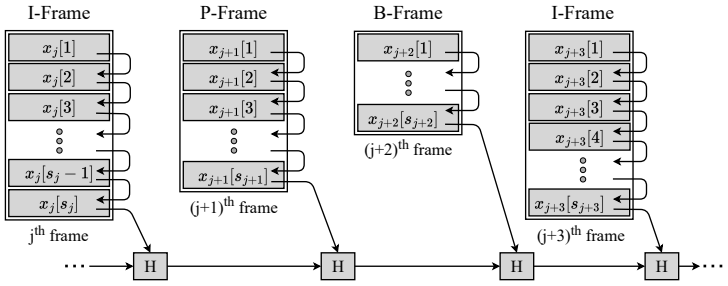

Figure 2: Data flow of the cumulative hash H_Φ using standard block hashes. For each element x_i, we denote its size as $s_i = |x_i|$.

Maintaining only the sequential hash of data is compact but inefficient for authenticating arbitrary disclosures (e.g. trims). Specifically, to authenticate a trim $\mathcal{T} = \{x_k, \ldots, x_{k+m}\}$ from $X = \{x_1, \ldots, x_n\}$, the prover must traverse all elements from x_k to x_n in order to reconstruct the chain from ℓ_x^k to ℓ_x^n. This inefficiency arises because there is no direct link between ℓ_x^k and ℓ_x^n without processing the entire intermediate sequence. To address this limitation, a VC must be constructed over X so that any element can be efficiently and independently authenticated [CF13]. Recent work [PSG+24, TB23] introduces VC schemes based on optimized tree structures that offer efficient updates and proofs.

In our setting, we adopt a minimal design based on a standard Merkle tree to avoid additional complexities. We extend the tree construction with a chaining mechanism: each leaf is computed by hashing its own value together with the cumulative hash of the previous element. Such a structure can be naturally instantiated over the sequence hasher from Definition 6, as shown in Figure 3. This approach enables efficient trim authentication: only a single Merkle proof for the last element of the trim is required, along with the hash chain linking the start and end of the trimmed segment.

Definition 7. Given sequence $h_x^* = \{h_x^1, ..., h_x^n\}$, with sequential chained hash ℓ_x^n, we construct a Merkle tree over it as a quadruple PPT algorithms $(\mathsf{KeyGen}_\Delta, \mathsf{Com}_\Delta, \mathsf{Open}_\Delta, \mathsf{Vfy}_\Delta)$:

- $pp \leftarrow \mathsf{KeyGen}_\Delta(1^\lambda, n)$: The generator outputs public parameters pp based on the security parameter λ, and vector size n.
- $\Delta_X \leftarrow \mathsf{Com}_\Delta(pp, h_x^*)$: Commits to all elements in the sequential data X according to the structure illustrated in Figure 3.
- $\delta_k \leftarrow \mathsf{Open}_\Delta(pp, \ell_x^k, k, h_x^k, \ell_x^{k-1})$: Produces a proof (in our case, a Merkle path/proof) for opening the commitment in the k^{th} element in the sequence, where $k \in [n]$ and $\ell_x^k = H(\ell_x^{k-1}||h_x^k)$.
- $b \leftarrow \mathsf{Vfy}_\Delta(pp, \Delta_X, \ell_x^k, k, \delta_k)$: The verifier accepts the proof δ_k if and only if $\ell_x^k = H(\ell_x^{k-1}||h_x^k)$ is indeed the k^{th} leaf of the tree with root Δ_X.

Theorem 1. *Assuming the hash functions H and H_ϕ are collision-resistant, and the underlying zkSNARKs is knowledge-sound; the proposed Merkle tree construction in Def. 7 satisfies position-binding.*

Sketch. Consider the commitment Δ_X and two elements, x_k and x'_k, in the sequence X. For successful verification through Vfy_Δ, the adversary $\mathcal{A}$ must be able to open both values at the same position. The probability of the adversary's success is calculated as follows:

$$\Pr\left[\begin{array}{c}\mathsf{Vfy}_\Delta(\Delta_X, \ell_x^k, k, \delta_k) \wedge \mathsf{Vfy}_\Delta(\Delta_X, \hat{\ell}_x^k, k, \hat{\delta}_k) \\ \wedge (h_x^k \neq \hat{h}_x^k \vee x_k \neq x'_k)\end{array}\middle|\begin{array}{c}pp \leftarrow \mathrm{Gen}(1^\lambda, \mathrm{n}) \\ \Delta_X, \ell_x^k, \hat{\ell}_x^k, \delta_k, \hat{\delta}_k, x_k, x'_k \leftarrow \mathcal{A}\end{array}\right]$$

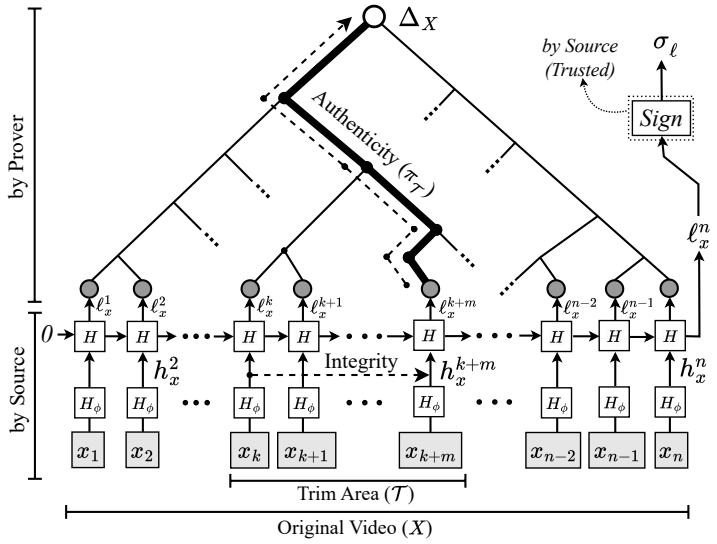

Figure 3: Proposed VC scheme for sequential data.

This could occur in two scenarios. First, $\mathcal{A}$ may find two preimages for the hash function H_ϕ, which is assumed to be collision-resistant under the PPT adversary model. Second, $\mathcal{A}$ could successfully open the leaf ℓ_x^k with two different values, i.e., $H(\ell_x^{k-1}\|h_x^k) = H(\hat{\ell}_x^{k-1}\|\hat{h}_x^k) \wedge h_x^k \neq \hat{h}_x^k$. This would imply that the adversary has found a collision in H, which occurs with negligible probability. Consequently, the overall probability of a successful attack (above probability) is negligible, and the proposed VC satisfies *position-binding.* □

4.2 Proving Construction of Δ_X from ℓ_x^n

Here, we demonstrate how to efficiently prove the trustless construction of a Merkle tree with root Δ_X based on the committed hash values of sequential data, i.e., $(\ell_x^i)_{i\in[n]}$.

Definition 8. Let X be the original sequence with elements $\{x_1, ..., x_n\}$ and chained hash values $(\ell_x^i)_{i\in[n]}$, where the final hash ℓ_x^n is authenticated by a signature σ_ℓ. Let Δ_X denote the root of a Merkle tree constructed over the set $L = (\ell_x^i)_{i\in[n]}$, where each ℓ_x^i is computed as $H(\ell_x^{i-1}||h_x^i)$, with $\ell_x^0 = 0$. We define zkSNARKs relation $\mathcal{R}_{\mathsf{VC}}$ for the public statement $\mathbf{st} = (\ell_x^n, \Delta_X)$ and provate witness $\mathbf{wit} = ((\ell_x^i)_{i\in[n]}, (h_x^i)_{i\in[n]}, (\delta_i)_{i\in[n]})$, for constructing the Merkle tree with respect to ℓ_x^n as follows:

$$\mathcal{R}_{\mathsf{VC}} := \big\{\big(\mathbf{st}: (\ell_x^n, \Delta_X),\ \mathbf{wit}: ((\ell_x^i)_{i\in[n]}, (h_x^i)_{i\in[n]}, (\delta_i)_{i\in[n]})\big) : \ell_x^0{=}0 \wedge \forall i{\in}[n], \ell_x^i{=}H(\ell_x^{i-1}\|h_x^i) \wedge \mathsf{Vfy}_\Delta(\Delta_X,\ell_x^i,i,\delta_i){=}1\big\} \quad (1)$$

Theorem 2. *Assuming the hash functions H and H_ϕ are collision-resistant, and the underlying zkSNARKs is knowledge-sound; the probability of a PPT adversary breaking soundness of $\mathcal{R}_{\mathsf{VC}}$ is negligible.*

sketch. We refer to the sent Merkle root from adversary as Δ'_X and the signed commitment ℓ_x^n with the signature σ'_ℓ. Moreover, we assume that the sequence X consists of n elements $\{x_1, \ldots, x_n\}$. Overall, we calculate the probability of the adversary $\mathcal{P}^*$ being able to

generate false proofs of the statement in Definition 8 as follows:

$$\Pr\left[\begin{array}{c} \Big(\exists\, i, \big(\,\mathsf{Vfy}_{\Delta}(\Delta_X, z'_i, i, \hat{\pi}^i_{\Delta}) = 1 \;\wedge (\delta_i \neq \hat{\delta}_i \;\vee\; z_i \neq z'_i)\big) \\ \vee\big(z_i = H(z_{i-1}\|\hat{h}^i_x) \wedge \hat{h}^i_x \neq h^i_x\big) \;\vee\; \big(z_0 = 0 \wedge z_n \neq \ell^n_x\big)\Big), \\ \exists\, i,\; \mathsf{Vfy}_{\Delta}(\Delta_X, z_i, i, \delta_i) \neq 1,\; \mathsf{Vfy}(vk, n, z_0, z_n, \Pi) = 1 \end{array} \middle| \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda),\; (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ (\delta_i, z_i)_{i\in[n]}, \Pi \leftarrow \mathcal{P}^* \\ (\omega_1, \ldots, \omega_n) \leftarrow \mathcal{E}(pp, z_0, z; \rho) \\ \forall\, i{\in}[n]:\;\; (h^i_x, \delta_i) \leftarrow \omega_i \;\;\wedge\;\; z_i \leftarrow H(z_{i-1}\|h^i_x) \end{array}\right]$$

The argument proceeds in three parts: (1) If the adversary submits a valid proof with correct public parameters but finds a $\hat{h}^i_x{\neq}h^i_x$ such that $H(z_{i-1}\|h^i_x) {=} H(z_{i-1}\|\hat{h}^i_x)$, then the adversary has found a collision in H. Due to the collision resistance of H, this occurs with negligible probability. (2) Another possibility is that the adversary successfully opens a leaf in Δ_X using two different Merkle paths, $\delta_i \neq \hat{\delta}_i$, or with two distinct leaf values, $z_i \neq z'_i$. Both of these scenarios would require the adversary to find collision in H because the entire Merkle path is simply a chained application of H. (3) Alternatively, the adversary could attempt to verify a malformed proof Π' using public parameters where $z_n \neq \ell^n_x$, while ensuring $z_0 = 0$. This would require breaking the soundness property of the employed IVC scheme. Therefore, the overall probability of a PPT adversary successfully breaking the soundness of the proof π_Δ is negligible. □

5 Protocol Details

In this section, we present our three-phase protocol (1. *Commit*, 2. *Prove*, and 3. *Verify*) for trustlessly authenticating arbitrary trims of sequential data without revealing the original source. Figure 4 provides detailed steps that each actor in the proposed protocol takes.

Commitment Phase: During this phase, the source (e.g., a camera or a trusted application like TruePic or ProvCam [LYC+24]) commits to the captured sequential data (e.g., video) using the hash algorithm defined in Definition 6, as depicted in Figure 3. To finalize the commitment, the device signs the computed commitment hash ℓ^n_x and outputs both the recorded data and the corresponding digital signature σ_ℓ. We argue that this commitment process is computationally feasible on existing hardware, given that standard cryptographic hash functions such as SHA-256 are already supported in commercial recording devices [son22].

Prove Phase: This phase consists of three steps: (1) The prover first proves construction of the full VC as detailed in §4. (2) Next, the prover proves arbitrary trims of the sequence using the authenticated Δ_X. (3) Optionally, the prover may choose not to reveal the original signature and instead anonymize the signer's identity [DEH25].

I) Prove $\boldsymbol{\Delta_X \sim \ell^n_x}$: This step is required only once per dataset. It involves constructing a Merkle tree from the recorded data outside the device using a ZK circuit. The circuit implements the statement defined in Definition 8, proving that the Merkle tree is correctly derived while computing the leaf hashes, as illustrated in Figure 3. The resulting proof establishes a verifiable link between Δ_X and ℓ^n_x, ensuring that Δ_X is authenticated with respect to a valid signature σ_ℓ for ℓ^n_x. This Merkle-based structure enables flexible selection of arbitrary start and end points when trimming the sequence.

II) Prove $\boldsymbol{\mathcal{T} \in \mathbf{X}}$: Once Δ_X is established, prover generates a proof to validate authenticity and integrity of trimmed segment. The prover must prove following statement.

Definition 9. Let X be the original sequence consisting of n elements $x_1, ..., x_n$, and let Δ_X be the VC corresponding to ℓ^n_x. We define the zkSNARKs relation $\mathcal{R}_{\mathsf{trim}}$ for the pulic statement $\mathbf{st} = (\Delta_X, \ell^{k+m}_x)$ and private witness $\mathbf{wit} = (\delta_{k+m})$, attesting that a trim $\mathcal{T}$ ending with cumulative hash ℓ^{k+m}_x (starting at the k-th element with a length of m) is

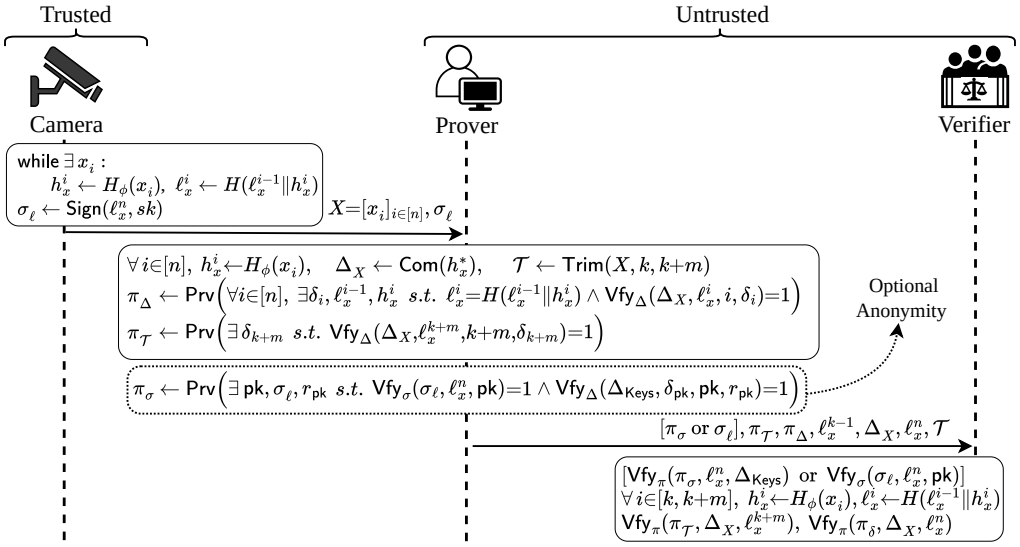

Figure 4: Overview of the protocol.

indeed an authentic trim of X, as follows:

$$\mathcal{R}_\mathsf{trim} := \big\{\big(\textbf{st:}\ (\Delta_X, \ell_x^{k+m}),\ \ \textbf{wit:}\ (\delta_{k+m})\big) : \mathsf{Vfy}_\Delta(\Delta_X, \ell_x^{k+m}, k{+}m, \delta_{k+m}) = 1\big\} \tag{2}$$

Theorem 3. *Assuming the hash functions H and H_ϕ are collision-resistant, and the underlying zkSNARKs is knowledge-sound; the probability of a PPT adversary breaking soundness of $\mathcal{R}_\mathsf{trim}$ is negligible.*

sketch. Let $\mathcal{T}'{=}\{x'_k, ..., x'_{k+m}\}$ be the trimmed sequence from the adversary and Δ_X the corresponding commitment. Given original sequence X, we aim to compute the probability of adversary $\mathcal{P}^*$ generating false proofs for the statement in Def. 9 as below.

$$\Pr\left[\begin{array}{c} \mathsf{Vfy}_\pi(vk, u, \pi_\mathcal{T}) = 1, \\ \mathsf{Vfy}_\Delta(\Delta_X, \ell_x^{k+m}, k{+}m, \delta_{k+m}){\neq}1 \end{array} \middle| \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda),\ (pk, vk) \leftarrow \mathcal{K}(pp, H),\ (\mathcal{T}', \delta_{k+m}, \ell_x^{k+m-1}, h_x^{k-1}) \leftarrow \mathcal{P}^* \\ \forall i{\in}[k, k{+}m] : h_x^i = H_\phi(x_i),\ \ell_x^{k+m} \leftarrow H(\ell_x^{k+m-1}||h_x^{k+m}) \end{array}\right]$$

Arguing that the above probability is negligible in the PPT model is straightforward. This follows from the collision resistance of the hash function H, H_ϕ and the soundness of the underlying zkSNARKs. Our argument proceeds in three parts: (1) Let's assume the adversary submits a valid proof with valid public parameters but manages to find an $x'_i \neq x_i$, such that $H_\phi(x'_i){=}H_\phi(x_i){=}h_x^i$ (Figure 5). Thus, the adversary must have found a collision in H_ϕ, which has a negligible probability due to the collision resistance property of it. (2) Another scenario is that adversary manages to find a collision in H for calculating $H(\ell_x^{k+m-1}||h_x^{k+m})$, which is not possible in our PPT adversary model. (3) Alternatively, the adversary must manage to successfully verify a malformed proof $\pi'_\mathcal{T}$ using some public parameters as $\hat{\ell}_x^{k+m} \neq \hat{\ell}_x^{k+m}$. The probability of this for a PPT adversary is negligible according to the soundness property of the employed zkSNARKs. Therefore, the overall probability of a PPT adversary breaking the soundness of $\pi_\mathcal{T}$ is negligible. □

III) [Optional] Anonymizing Signer's Identity: This optional protocol section is similar to the constructions on prior work [DEH25, DCB24, LGS+26], which can be directly applied to our framework as well. To anonymize the original signer's identity, we assume existence of a public VC to a known set of verified public keys, established during

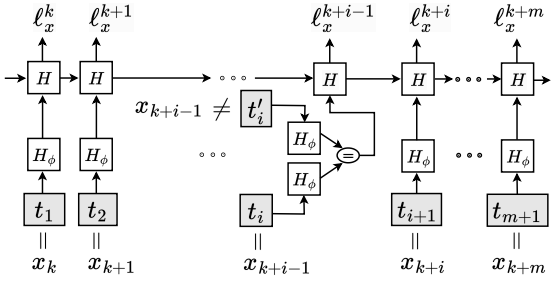

Figure 5: Breaking *position-binding* of our VC, reduces to finding two preimages for H_ϕ.

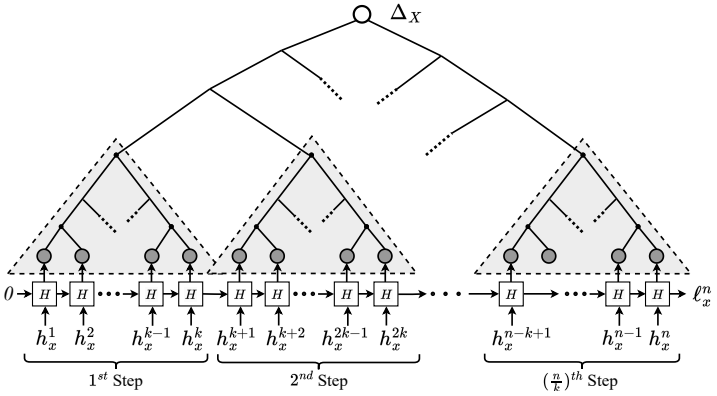

Figure 6: Proving the relation $\mathcal{R}_{\mathsf{VC}}$ in a folding-based setup. The proof has $\frac{n}{k}$ steps, each handling k leaves (subtrees of height $\log k$) and asserting the Merkle inclusion in Δ_X.

a secure ceremony with trusted authorities. Instead of revealing the signature σ_m for the message m, the prover generates a zkSNARK proof using the following definition.

Definition 10. Assume that all verified public keys are committed to by a VC scheme with public value of Δ_{PK}. Let verification function Vfy_σ such that $\mathsf{Vfy}_\sigma(m, \sigma_m, pk) = 1$ if and only if σ_m is a valid signature for m under the public key pk. We define $\mathcal{R}_{\mathsf{sig}}$ for public statement $\mathbf{st} = (m, \Delta_{PK})$ and with the private witness $\mathbf{wit} = (\sigma_m, \delta_i, pk_i)$ as follows:

$$\mathcal{R}_{\mathsf{sig}} := \big\{ \big(\mathbf{st}\text{: } (m, \Delta_{PK}),\ \ \mathbf{wit}\text{: } (\sigma_m, \delta_i, pk_i)\big) : \\ \mathsf{Vfy}_\sigma(m, \sigma_m, pk_i) = 1 \ \wedge\ \mathsf{Vfy}_\Delta(\Delta_{PK}, pk_i, i, \delta_i) = 1 \big\} \quad (3)$$

Verification Phase: During this phase, the verifier must check at least two proofs: π_Δ and $\pi_{\mathcal{T}}$. Additionally, they need to verify either the signature σ_ℓ or a proof of its existence, π_σ. Notably, once π_Δ and either σ_ℓ or π_σ are verified, there is no need to reverify them for future trims of the same sequence.

6 Implementation Details

In this section, we describe the implementation of the protocol's proving phase, focusing on the relations $\mathcal{R}_{\mathsf{VC}}$ and $\mathcal{R}_{\mathsf{trim}}$. The optional signer-anonymization relation, $\mathcal{R}_{\mathsf{sig}}$, has been extensively studied and formalized in prior work, including [LGS+26]. We therefore omit its implementation details for brevity.

6.1 Proving $\Delta_X \sim \ell_x^n$

Choice of proof system. Proving $\mathcal{R}_{\mathsf{VC}}$ requires proving correct VC construction in ZK, which is computationally demanding, primarily due to memory requirements. As observed in prior work [DCB24], even for relatively small data sequences, the memory consumption of general-purpose zkSNARK backends grows rapidly, making proof generation impractical on commodity hardware. We address this challenge by observing that VC construction is inherently recursive, consisting of repeated applications of a hash function. For example, a Merkle tree of height 4 (16 leaves) can be viewed as a height-2 tree whose leaves are themselves roots of height-2 subtrees. This recursive structure naturally enables the use of folding-based zkSNARKs, such as Nova [KST22], where each folding step must execute identical computations. By batching hash operations to fit this uniform structure, we significantly reduce the memory and computational overhead of proving long sequences. In particular, the peak memory usage is bounded by the maximum memory required by an individual folding step rather than by the entire computation.

Figure 6 illustrates the process of proving the construction of a Merkle tree with root Δ_X from a sequence of hashes, where the final hash value corresponds to ℓ_x^n. Algorithm 1 outlines the computations performed at each folding step within our IVC setup. Specifically, we partition the Merkle tree into k subtrees (Figure 6), and prove that each subtree is part of the original tree by providing witness to the existence of a path from the root of the subtree to Δ_X. The arrays $[p_i]$ and $[s_i]$ in the inputs represent the sibling values and their corresponding positions within the Merkle path, respectively. Additionally, in each step, we compute the cumulative hash of the subtree's leaves and propagate the final result to the subsequent step as part of the public output.

We used Nova library written in `Rust` for implementation as described in next section. Nova provides one of the fastest provers in the literature, with its cost dominated by two $O(|F|)$ multiexponentiations and no FFTs, making it more efficient than SNARK-based baselines [KST22]. It also achieves low recursion overhead via a constant-sized verifier circuit (approximately 10,000 multiplication gates) [nov24a]. As an incremental proof system, Nova (and its generalization SuperNova) provides significant improvements in memory usage compared to non-incremental baselines [KS22].

Algorithm 1: Folding Step i

Aux. In : $[h_x^{(i-1)k+1}, \ldots, h_x^{i\times k}]$, $[p_1, \ldots, p_{\log(\frac{n}{k})}]$, $[s_1, \ldots, s_{\log(\frac{n}{k})}]$
Pub. In : Δ_X, $\ell_x^{(i-1)k}$, **Pub. Out :** Δ_X, $\ell_{i\times k}$

1 **for** $j : 1 \to k$ **do:** $\ell_x^{(i-1)k+j} \leftarrow H(\ell_x^{(i-1)k+j-1} || h_x^{(i-1)k+j})$;
2 $node \leftarrow \text{buildTree}([\ell_x^{(i-1)k+1}, \ldots, \ell_x^{i\times k}])$;
3 **for** $j : 1 \to \frac{n}{k}$ **do**
4 **if** $s_j == 0$ **then:** $node \leftarrow H(node || p_j)$; **else:** $node \leftarrow H(p_j || node)$;
5 **assert** $\Delta_X == node$;
6 **output** Δ_X, $\ell_x^{i\times k}$;

6.2 Proving $\mathcal{T} \in \mathbf{X}$

Choice of proof system. In contrast to $\mathcal{R}_{\mathsf{VC}}$, proving $\mathcal{R}_{\mathsf{trim}}$ is considerably simpler, as it only requires proving correct execution of $\mathsf{Vfy}_\Delta(\Delta_X, \ldots)$=1. In our Merkle tree instantiation, this amounts to proving a private Merkle inclusion path consisting of $\log n$ hash evaluations. Since this is a standard construction with modest memory and computational requirements, we implement it in `Circom` using Groth16 as the proving

backend. However, any generic zkSNARKs system can be used.

Algorithm 2 outlines computations necessary for this phase. As discussed in Section 5, proving the hash of the trim $\mathcal{T}=\{x_m, \ldots, x_{m+k}\}$ itself is not required when the trim is provided to the verifier. This is because the verifier can independently compute the cumulative hash of the trim elements to derive h_x^{m+k} locally. Consequently, the prover only needs to show that h_x^{m+k} is a valid opening of a leaf in the Merkle tree rooted at Δ_X (see Definition 9). Notably, this makes the proof complexity independent of the trim size.

Algorithm 2: Proving $\mathcal{T} \in X$

Aux. In : ℓ_x^{m-1}, $[p_1, \ldots, p_{\log n}]$, $[s_1, \ldots, s_{\log n}]$, **Pub. In :** Δ_X, h_x^j

1 $node \leftarrow H(\ell_x^{i-1} || h_x^j)$;
2 **for** $i : 1 \rightarrow \log n$ **do**
3 ⌊ **if** $s_i == 0$ **then:** $node \leftarrow H(node || p_i)$; **else:** $node \leftarrow H(p_i || node)$;
4 **assert** $\Delta_X == node$;

6.3 Verification

In this phase, the verifier must verify π_Δ and $\pi_\mathcal{T}$ as described in Algorithm 3. Additionally, the verifier should verify either the signature σ_ℓ or a proof of its existence, π_σ (Definition 10). The algorithm begins by authenticating ℓ_x^n in line 1. Next, it verifies Δ_X with respect to the previously authenticated ℓ_x^n in line 2. Finally, the verifier authenticates the received trim $\mathcal{T}$ by computing its cumulative hash and verifying $\pi_\mathcal{T}$ using the already verified Δ_X and the locally computed hash value as public inputs (lines 3 and 4).

Algorithm 3: Verification

Pub. In : $\mathcal{T}$, Δ_X, π_Δ, $\pi_\mathcal{T}$, ℓ_x^n, (σ_ℓ or π_σ), ℓ_x^{k-1}
Pub. Out : *true* or *false*

1 **assert** $\mathsf{Vfy}_\sigma(\ell_x^n, \sigma_\ell) == 1$ *or* $\mathsf{Vfy}_\pi(\ell_x^n, \pi_\sigma) == 1$;
2 **assert** $\mathsf{Vfy}_\pi(\Delta_X, \pi_\Delta, \ell_x^n) == 1$;
3 $temp \leftarrow \ell_x^{k-1}$;
4 **for** $i : 1 \rightarrow |\mathcal{T}|$ **do:** $temp \leftarrow H(temp || H_\phi(t_i))$;
5 **assert** $\mathsf{Vfy}_\pi(\Delta_X, \pi_\mathcal{T}, temp) == 1$;

7 Performance Evaluation

Reproducibility. Appendix A provides detailed information regarding reproducibility of the benchmarks. All reported benchmarks are reproducible using scripts and `README.md` file provided in our repository[6], containing full implementation of our protocol.

7.1 System Setup

Table 3 summarizes our experimental environment. To evaluate the practicality of our constructions, we benchmarked our implementation on a mid-range *ASUS Zephyrus G14* laptop with an AMD Ryzen CPU and 16 GB of RAM running *Fedora 44 Workstation*, representing the prover's main device. The source device is a *Raspberry Pi Zero 2W* with

[6] https://github.com/zero-savvy/proven-view

Table 3: Experimental Setup Configuration

Device	CPU			Mem.	SSD	OS
	Model	Freq.	Cores			
ASUS Zephyrus G14	AMD Ryzen® 9 6900HS	3.3∼4.9 GHz	8 (16)	16 GB	512 GB	Fedora 44
Rspbry. Pi Zero 2W	Arm Cortex A53	1 GHz	4	512 MB	16 GB	Ubuntu 22.04

Table 4: Different Video Configurations

Category	Security Camera			Dashcam			Professional (News, Live TV)		
Res.✪	FHD	4K		FHD		4K	FHD		4K
FPS	30			30	60		50/60	24	30/60
Comp.	H.264		H.265	H.264			MPEG-2/H.264	ProRes 422	H.265
Bitrate*	2 Mbps	8 Mbps	4 Mbps	10 Mbps	20 Mbps	30 Mbps	20 Mbps	220 Mbps	20 Mbps
MB/min*	∼15 MB	∼60 MB	∼30 MB	∼75 MB	∼150 MB	∼225 MB	∼150 MB	∼1500 MB	∼150 MB
Time/GB*	∼67 mins	∼17 mins	∼33 mins	∼13 mins	∼7 mins	∼4.5 mins	∼7 mins	<1 min	∼7 mins

✪ FHD: 1920×1080 and 4K: 3840×2160 * Estimated

512 MB of RAM and a 16 GB SD card, running *Ubuntu 22.04 Server*, responsible for generating and authenticating the original sequence.

We emphasize that even the *Raspberry Pi Zero 2W* is substantially more capable than commodity TPM chips. Our goal is to demonstrate that even an SoC reaches its limits when performing VC construction during the commitment phase. At the same time, the trust assumptions and resistance to physical attacks provided by an SoC are not comparable to those of a tamper-resistant TPM chip. As discussed in § 3.1, TPM chips are highly constrained and support only a limited set of cryptographic primitives with much lower performance than typical crypto-microprocessors used in SoC platforms.

7.2 Commitment Phase

We first analyze the computational and storage complexity of the two steps in our VC construction: the sequence hash (Definition 6) and the tree-based structure on top of it (Definition 7). As discussed earlier, the commitment phase must remain lightweight to sustain high throughput and minimize delays in real-time processing. Here, we quantify the overhead of constructing a full VC on a resource-constrained device.

Table 4 summarizes approximate bitrates for different video configurations, ranging from minimal FHD 30 fps security camera footage to 4K 60 fps broadcast sources. Depending on quality and motion levels, bitrates vary from 2 Mbps to over 100 Mbps. Figure 7a shows the approximate number of hashes required per second to digest the media at different bitrates, assuming 256-bit blocks. To evaluate feasibility, we benchmarked our constrained device (i.e. *Raspberry Pi Zero 2W*) against several hash algorithms, as shown in Figure 7b. For standardized hash functions such as SHA, we used OpenSSL's implementations, which leverage cryptographic acceleration available on the Cortex-A53 SoC. In contrast, ZK-friendly hash functions such as Poseidon do not benefit from hardware support on such boards. The most efficient implementation we found is a C library [pos25], yet even for low bitrates (e.g., 2 Mbps), hashing one second of media with Poseidon exceeds one second of computation, making real-time processing infeasible. This indicates constrained devices cannot sustain both sequence hashing and additional VC layer when relying on Poseidon.

Table 5 compares the computational cost of the commitment phase when executed on the constrained device versus a more capable machine (e.g., a midrange laptop). On the laptop, constructing a complete VC over long footage (up to 6 hours) requires only a few minutes. On the constrained device, the same operation can exceed one hour of continuous computation. Importantly, the device must also compute H_Φ over the entire footage. With Poseidon, this would take more than 10 hours for 30 minutes of footage and hundreds of hours for 6 hours of footage. By contrast, standardized hash functions with hardware acceleration on devices such as the *Raspberry Pi Zero 2W* can compute the same task in minutes (e.g., 1∼5 minutes for 30 minutes of footage). This suggests

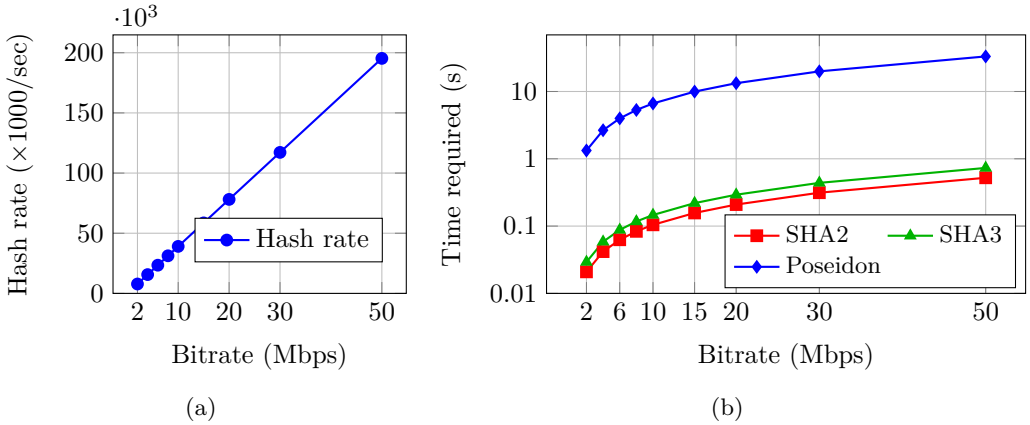

Figure 7: (a) Hash rate for different bitrates. (b) Log-scale comparison of time required by the source device to calculate total hashes of a 1-second video at different bitrates.

that constrained devices can feasibly compute chained sequence hashes using standardized functions during capture, consistent with prior work [LYC+24].

We further provide an estimate of the minimum memory required to update a VC with newly generated frames every second under an idealized computation model that assumes zero-overhead access to external storage. This represents a best-case lower bound and excludes the additional memory required for computing hashes of incoming frames. The goal is to illustrate that, even under these idealized assumptions, the memory required solely for maintaining the VC already approaches or exceeds the capacity limits of current TPM standards (Table 1). In practice, at least twice this amount of memory is typically required to maintain separate read and write buffers for efficient operation. In contrast, our proposed approach only requires the device to maintain a single cumulative hash over the frame stream. As a result, the device stores only approximately $2 \times 32 = 64$ bytes of state, which is well within typical TPM constraints.

7.3 Proving Phase

Choice of hash functions. The main motivation for outsourcing VC construction is to enable practical construction of a ZK-friendly VC that can be directly used by schemes such as [DEH25, DCB24, ZYO+24]. Accordingly, we benchmark VC construction using Poseidon [GKR+21]. In contrast, media hashing, i.e., the computation of H_ϕ over individual frames to obtain the h_x^i values, is performed entirely on the source device and is not part of $\mathcal{R}_{\mathsf{VC}}$, $\mathcal{R}_{\mathsf{trim}}$, or $\mathcal{R}_{\mathsf{sig}}$. Therefore, we use SHA-family primitives for this purpose, as these are widely supported by constrained devices. For computing the cumulative hashes ℓ_x^i, which are also generated by the source device, we consider two scenarios: (1) SHA2 and (2) Poseidon. The first scenario is the most compatible with existing TPM standards. However, it increases the complexity of proving VC construction (§ 6.1), since the prover must additionally prove the SHA-based leaf chain, even though the VC itself is constructed using Poseidon. The second scenario assumes that the trusted component in source device computes the cumulative chain using Poseidon. Since this operation is performed only at the frame rate, we argue that in some settings it is reasonable to assume that the device can sustain approximately 30–60 Poseidon evaluations per second. This substantially reduces the complexity of the proving circuit, as it eliminates SHA computations and allows the entire ZK circuit to rely on ZK-friendly hashes.

After receiving the initial commitment ℓ_x^n and its signature σ_ℓ from the device side, the prover constructs a Merkle tree based on the list of chained hash values and generates

Table 5: Commitment Complexity

			Sec. Cam		DashCam		News/TV	
Resolution			FHD	4K	FHD	4K	FHD	4K
Frame Rate			30		30	60	60	
Bitrate (Mb/S)			3	5	10	30	20	40
Hash/S in H_Φ			12 K	19 K	39 K	117 K	78 K	155 K
30 Minutes of Video	H_Φ time On R.Pi 0	SHA3	1.8 m	2.6 m	4.4 m	13 m	6.4 m	17 m
		SHA2	1.3 m	1.9 m	3.2 m	9.3 m	4.6 m	12 m
		POS	1.3 h	2 h	3.3 h	10 h	6.6 h	13 h
	VC calc. by R.Pi 0	SHA3	3.1 s			6.7 s		
		SHA2	2.2 s			4.8 s		
		POS	144 s			306 s		
	VC calc. by laptop	SHA3	< 1 s			< 1 s		
		SHA2	< 1 s			< 1 s		
		POS	4.7 s			9.6 s		
	VC calc. minimum■ theoretical memory req.		$\lceil \log \texttt{total\_frames} \rceil = 16$ $2{\times}16{\times}32 + 2{\times}30{\times}32 \approx 3\,\text{KB}$			$\lceil \log \texttt{total\_frames} \rceil = 17$ $2{\times}17{\times}32 + 2{\times}60{\times}32 \approx 4\,\text{KB}$		
6 Hours of Video	H_Φ time On R.Pi 0	SHA3	21 m	29 m	52 m	2.8 h	2.1 h	3.5 h
		SHA2	15 m	21 m	37 m	2 h	1.5 h	2.5 h
		POS	16 h	24 h	38 h	120 h	80 h	160 h
	VC calc. by R.Pi 0	SHA3	46.8 s			99.1 s		
		SHA2	33.4 s			70.3 s		
		POS	35 m			75 m		
	VC calc. by laptop	SHA3	< 1 s			$\sim$2 s		
		SHA2	< 1 s			$\sim$2 s		
		POS	$\sim$1 m			$\sim$2 m		
	VC calc. minimum■ theoretical memory req.		$\lceil \log \texttt{total\_frames} \rceil = 20$ $2{\times}20{\times}32 + 2{\times}30{\times}32 \approx 3.5\,\text{KB}$			$\lceil \log \texttt{total\_frames} \rceil = 21$ $2{\times}21{\times}32 + 2{\times}60{\times}32 \approx 5\,\text{KB}$		

■ Minimum memory required to update VC with newly generated frames every second under an idealized computation model assuming zero-overhead access to external storage. This represents a best-case lower bound and illustrates that, even under these idealized assumptions, the memory required solely for maintaining the VC already approaches/exceeds limits of current TPM standards (Table 1). * *s*, *m*, and *h* in the times indicate *seconds*, *minutes*, and *hours*, respectively.

a proof to attest its correctness. To enhance proof generation efficiency, we employ the Nova folding scheme [KST22] (refer to § 6.1 for details). Our implementation utilizes the `nova-snark` [nov24a] backend and the `nova-scotia` front-end [nov24b] to translate our defined steps in `Circom` [BMIMT+22] into Nova. `Circom` is a widely adopted DSL for ZK applications, used to construct arithmetic circuits. It compiles a program into a witness generator, which computes intermediate values, and a Rank-1 Constraint System (R1CS), which defines the arithmetic circuit used by zkSNARK provers and verifiers [BMIMT+22, WSC+24]. A practical security analysis of Circom circuits has been provided in [WSC+24].

As illustrated in Figure 6, we optimize the Merkle tree construction by dividing it into subtrees. Table 6 presents detailed performance benchmarks for different sequence sizes. To determine the optimal subtree size proven in each step of the folding, we generated proofs for various subtree sizes while keeping the overall tree size fixed. As shown in Table 6, for a Merkle tree of height 20, increasing the subtree size reduces proving time but increases memory usage. To the best of our knowledge, this is the first benchmark for verifiable Merkle tree reconstruction within a ZKP, leveraging folding schemes.

Once the Merkle tree is constructed, the trim range can be chosen flexibly. The complexity of the proof $\pi_{\mathcal{T}}$ remains independent of the trim's size or position because it only proves the valid opening of the commitment corresponding to the last element of the trim, establishing its inclusion in our VC with a valid path to the root Δ_X. For this proof, we can employ any general zkSNARK framework. We report performance metrics for generating *Groth16* [Gro16] proofs for this step, but alternative zkSNARKs such as Spartan [Set20] can also be used[7].

[7] We note that Groth16 is not a transparent zkSNARK and therefore requires a trusted setup that depends on the underlying circuit. This setup must be performed correctly in real-world deployments to provide the intended security guarantees [Gro16]. We do not consider this aspect further, as it does

Table 6: Performance Analysis

video format	Commitment			$\mathcal{R}_{\mathsf{VC}}$: $\pi_\Delta \leftarrow \mathcal{P}(\Delta_X \sim \ell_x^n)$				$\mathcal{R}_{\mathsf{trim}}$: $\pi_\mathcal{T} \leftarrow \mathcal{P}(\mathcal{T} \in X)$			
				Prover		Verifier		Prover		Verifier	
	tree height	subtree height	# of folds	time (m)	Mem. (GB)	time (s)	Proof (KB)	time (s)	Mem. (GB)	time (ms)	Proof (KB)
Scenario one: Using SHA-256 for both Frame Hash (h_x^i) and Chain (ℓ_x^i), Using Poseidon only for VC											
5-10 m @ 20 fps, 10-20 m @ 10-15 fps	13	3	2^10	8.7	1.7	0.5	∼10	<1 s	<1	∼50	0.8
		4	2^9	7.5	3.0	0.8	∼10				
		5	2^8	7.15	4.7	1.0	∼10				
10 m @ 60 fps, 20 m @ 30 fps	15	4	2^11	29.1	3.0	0.8	∼10	<1 s	<1	∼50	0.8
		5	2^10	26.5	4.7	1.1	∼10				
Scenario two: Using SHA-256 only for Frame Hash (h_x^i), but Poseidon for both Chain (ℓ_x^i) and VC											
10 m @ 60 fps, 20 m @ 30 fps	15	7	2^8	1.4	0.9	0.26	∼10	<1 s	<1	∼50	0.8
5 h @ 60 fps, 10 h @ 30 fps	20	8	2^12	36.6	1.4	0.5	∼10	<1 s	<1	∼50	0.8
		10	2^10	28.7	4.2	1.4	∼10				

A key observation is that proving the construction of the proposed VC for a long video footage (5 to 10 hours) takes less than 30 minutes on a midrange laptop, while proving multiple arbitrary trims of the sequence takes under a second. This indicates the practicality of the proposed scheme, while imposing minimal overhead to the original committer.

We also evaluate the cost of optional anonymization of the original signer. This is a straightforward construction, and existing ZK circuits [DEHM25] support proofs similar to our relation $\mathcal{R}_{\mathsf{sig}}$ in Definition 10. Our experiments show that proving possession of a signature under one of the public keys in a VC with 1024 members takes less than one second on a mid-range laptop, and verification completes within a few milliseconds.

7.4 Verification Phase

As illustrated in Figure 4, the verifier checks at least two proofs (π_Δ and $\pi_\mathcal{T}$), in addition to either a signature verification or its corresponding anonymized proof (π_σ). Table 6 reports the proof sizes and verification times for both π_Δ and $\pi_\mathcal{T}$ (corresponding to the last element of the trimmed sequence). We also measured peak memory consumption of approximately 0.7 GB and 0.4 GB during the verification of π_Δ and $\pi_\mathcal{T}$, respectively.

A key observation is the low computational overhead for the verifier and the compact proof size. For example, a trimmed video segment of several gigabytes produces a proof of only a few kilobytes, with a verification time of approximately 50 ms.

7.5 End-to-End Performance

We provide an end-to-end performance estimate of the proposed approach in two representative scenarios based on our benchmarks: (1) a 1-minute trim extracted from a 10-minute dashcam recording at 60 FPS, and (2) a 3-minute trim extracted from a 5-hour CCTV recording at 30 FPS.

For both scenarios, we assume that the source device uses SHA-256 for H_ϕ, i.e., computing individual frame hashes. For the cumulative hash chain, we consider two configurations: the dashcam scenario uses SHA-256, while the CCTV scenario uses Poseidon. The chain computation is performed at the frame rate, resulting in only 60 SHA-256 evaluations per second for the dashcam and 30 Poseidon evaluations per second for the CCTV example. This workload is feasible on our Raspberry Pi Zero 2W setup. However, we note that deploying Poseidon directly within highly constrained TPM chips would require additional hardware support or a software-based TEE, as existing TPM

not affect the proving or verification performance evaluated in this work, but we mention it here for completeness.

Table 7: End-to-end Performance Analysis

Scenario	Commit Phase ($\mathcal{S}$)	Prove Phase ($\mathcal{P}$)			Verify Phase ($\mathcal{V}$)
	Calc. hash chain ℓ_X	Calc. Δ_X	Prove π_Δ	Prove $\pi_\mathcal{T}$	Verify π_Δ & $\pi_\mathcal{T}$
DashCam FHD, 10 m @ 60 fps frame: SHA2 + chain: SHA2	∼1 min	< 1 s	∼26 min	< 1 s	< 1 s
CCTV FHD, 5 h @ 30 fps frame: SHA2 + chain: Poseidon	∼40 min	< 1 s	∼29 min	< 1 s	< 1 s

standards primarily support SHA-family primitives. We further require that the prover $\mathcal{P}$ constructs a Poseidon-based Merkle tree on top of the received chain from the source. Table 7 summarizes the resulting end-to-end performance. Note that the commitment phase is performed during the recording, which means the reported numbers are scattered over the entire recording duration and do not represent a single blocking operation.

8 Discussion and Future Work

Summary. We present a method for practically delegating VC construction in resource-constrained settings, targeting privacy-preserving partial disclosures of different type of time series. The main advantage of our approach is the minimal computational and storage overhead imposed on the source device, which is often constrained, while keeping the prover's workload within practical bounds. Our method only relies on computing a chained cumulative hash over the sequence elements at the source. Any party with access to these chained hashes can later (re)construct the VC and prove the authenticity of arbitrary trims. Importantly, the complexity of verifying a trim remains constant, independent of the trim length or the underlying data complexity. Furthermore, To achieve efficient ZKPs for full VC construction on the prover side, we leverage folding-based zkSNARKs, i.e. Nova [nov24a]. We implemented the protocol as an open-source library, available on GitHub, and evaluated its performance using a *mid-range laptop* as the *untrusted prover* and a *Raspberry Pi Zero 2W* as the resource-constrained *source device*. The results confirm the practicality of offloading VC construction in realistic scenarios.

Limitations. The protocol is agnostic to media encoding formats and treats frames as abstract byte strings. As a result, the construction provides byte-exact provenance rather than perceptual provenance. Consequently, any modification to the underlying byte representation, such as re-encoding with a different compression method, results in different hashes and invalidates the original signature. Therefore, trimming and disclosure operations must preserve the original byte representation and be performed carefully to maintain compatibility with the authenticated source.

This further introduces a reconstructibility limitation (on the verifier side): when predictive encoding schemes (e.g., P-frames) are used, partial disclosures can be reconstructed only if trims begin at self-contained frames (e.g., I-frames, which are typically inserted at least once per second in CCTV standards) or if the media is normalized (e.g., through full-frame extraction). This ensures that frame hashes can be recomputed by the verifier.

Another limitation is that, although the construction prevents manipulation of frames within a single signed sequence, it does not enforce global ordering across independently signed sequences. One possible extension is to incorporate a trusted monotonic counter within the TPM and include its value in the signed message. This would enable proving that a trim derived from sequence X_1 was recorded before a trim derived from sequence X_2. However, this approach only applies to sequences generated by the same device/source and does not directly address multi-device scenarios. In such settings, additional trusted metadata, such as a trusted time source or global epoch mechanism, may be required, introducing further complexity into the overall system model.

Future Work. Future work will explore integrating recent advancements in folding techniques, optimizing VC schemes for even lower prover overhead and more efficient updates. Another interesting direction is to analyze the security implications of incorporating image hashing algorithms, such as *pHash* [SYG+25], within our framework, to potentially enhance perceptual integrity verification for visual data and decrease the computational complexity of chained hashes in the source device.

Acknowledgments

The authors thank the TCHES reviewing team for their thorough engagement and valuable feedback. The authors acknowledge the use of ChatGPT for grammar correction and improving clarity of the text. This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

References

[AB20] Alejandro Cabrera Aldaya and Billy Bob Brumley. When one vulnerable primitive turns viral: Novel single-trace attacks on ecdsa and rsa. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 196–221, 2020.

[app21] CSAM detection. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf, 2021.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*, pages 561–586. Springer, 2019.

[BMIMT+22] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.

[C2P24a] C2PA in DALL-E 3. https://help.openai.com/en/articles/8912793-c2pa-in-dall-e-3, 2024. Accessed: 2024-09-20.

[c2p24b] C2PA Technical Specification. https://c2pa.org/specifications/specifications/1.2/index.html, 2024. Accessed: 2024-01-03.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 55–72. Springer, 2013.

[CLZ23] Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang. Controlled data races in enclaves: Attacks and detection. In *32nd USENIX Security Symposium*, pages 4069–4086, 2023.

[CSB+26] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. TEE.fail: Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition . In *2026 IEEE Symposium*

on Security and Privacy (SP), pages 1877–1895, Los Alamitos, CA, USA, May 2026. IEEE Computer Society.

[DCB24] Trisha Datta, Binyi Chen, and Dan Boneh. Veritas: Verifying image transformations at scale. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 97–97. IEEE Computer Society, 2024.

[DEH25] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. Vimz: Private proofs of image manipulation using folding-based zksnarks. *Proceedings on Privacy Enhancing Technologies*, 2025.

[DEHM25] Stefan Dziembowski, Shahriar Ebrahimi, Parisa Hassanizadeh, and Susil Kumar Mohanty. TrafficProof: Privacy-preserving reliable traffic information sharing in social internet of vehicles. Cryptology ePrint Archive, Paper 2025/1062, 2025.

[EH24] Shahriar Ebrahimi and Parisa Hassanizadeh. From interaction to independence: zksnarks for transparent and non-interactive remote attestation. In *Network and Distributed Systems Security Symposium (NDSS)*, 2024.

[FCJ+23] Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22466–22477, 2023.

[FEYM24] Pierre Fernandez, Hady Elsahar, I Zeki Yalniz, and Alexandre Mourachko. Video seal: Open and efficient video watermarking. *arXiv preprint arXiv:2412.09492*, 2024.

[FXTB22] Jianwei Fei, Zhihua Xia, Benedetta Tondi, and Mauro Barni. Supervised gan watermarking for intellectual property protection. In *2022 IEEE WIFS*, pages 1–6. IEEE, 2022.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, volume 2021, 2021.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[GRWZ20] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. Pointproofs: Aggregating proofs for multiple vector commitments. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 2007–2023, 2020.

[HHZ+24] Kun Hu, Zixuan Hu, Qianhui Zhu, Xiaochao Wang, and Xingjun Wang. Stegavideo: Robust high-resolution video steganography with temporal and edge guidance. 2024.

[HLE+26] Jonas Hofmann, Philipp-Florens Lehwalder, Shahriar Ebrahimi, Parisa Hassanizadeh, and Sebastian Faust. Piranhas: Privacy-preserving remote attestation in non-hierarchical asynchronous swarms. In *Network and Distributed Systems Security Symposium (NDSS)*, 2026.

[JSS20] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy*, 18(2):56–60, 2020.

[KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.

[LGS+26] Shreyas Londhe, Aayush Gupta, Sora Suegami, Yogesh Shahi, Rute Figueiredo, Parisa Hassanizadeh, and Shahriar Ebrahimi. Zero-knowledge proofs of generalized regular expression matching for anonymized email verification. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2026.

[LNS+22] Yuxin Liu, Yoshimichi Nakatsuka, Ardalan Amiri Sani, Sharad Agarwal, and Gene Tsudik. Vronicle: verifiable provenance for videos from mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 196–208, 2022.

[LTLB24] Dongdong Lin, Benedetta Tondi, Bin Li, and Mauro Barni. A cyclegan watermarking method for ownership verification. *IEEE Transactions on Dependable and Secure Computing*, 2024.

[LYC+24] Yuxin Liu, Zhihao Yao, Mingyi Chen, Ardalan Amiri Sani, Sharad Agarwal, and Gene Tsudik. Provcam: A camera module with self-contained tcb for producing verifiable videos. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 588–602, 2024.

[MHE25] Piotr Mikołajczyk, Parisa Hassanizadeh, and Shahriar Ebrahimi. Towards trustless provenance: A privacy-preserving framework for on-chain media verification. *Cryptology ePrint Archive*, 2025.

[MJZ23] Souha Mansour, Saoussen Ben Jabra, and Ezzeddine Zagrouba. A robust deep learning-based video watermarking using mosaic generation. In *VISIGRAPP*, pages 668–675, 2023.

[nov24a] Nova high-speed recursive arguments from folding schemes. https://github.com/microsoft/Nova, 2024. Accessed: 2024-01-03.

[nov24b] nova-scotia. https://github.com/nalinbhardwaj/Nova-Scotia, 2024. Accessed: 2024-01-03.

[pos25] C and x86_64 implementation of Poseidon hash permutation. https://github.com/CryptoExperts/poseidon, 2025. Accessed: 2025-05-04.

[PSG+24] Charalampos Papamanthou, Shravan Srinivasan, Nicolas Gailly, Ismael Hishon-Rezaizadeh, Andrus Salumets, and Stjepan Golemac. Reckle trees: Updatable merkle batch proofs with applications. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1538–1551, 2024.

[Qua20] Snapdragon 888 5g mobile platform, 2020. Accesses: 2024-10-10.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

[SHNK22] Lukas Struppek, Dominik Hintersdorf, Daniel Neider, and Kristian Kersting. Learning to break deep perceptual hashing: The use case neuralhash. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 58–69, 2022.

[son22] Sony unlocks in-camera forgery-detection technology. https://www.sony.eu/presscentre/sony-unlocks-in-camera-forgery-proof-technology, 2022. Accessed: 2024-01-08.

[SYG+25] Shree Singhi, Aayan Yadav, Aayush Gupta, Shahriar Ebrahimi, and Parisa Hassanizadeh. Provenance detection for ai-generated images: Combining perceptual hashing, homomorphic encryption, and ai detection models. *arXiv preprint arXiv:2503.11195*, 2025.

[TB23] Ertem Nusret Tas and Dan Boneh. Vector commitments with efficient updates. *arXiv preprint arXiv:2307.04085*, 2023.

[tpm26] Tpm 2.0 library. https://trustedcomputinggroup.org/resource/tpm-library-specification/, 2026. Accessed: 2026-02-23.

[VSSY+24] Stephan Van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. Sok: Sgx. fail: How stuff gets exposed. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4143–4162. IEEE, 2024.

[WSC+24] Hongbo Wen, Jon Stephens, Yanju Chen, Kostas Ferles, Shankara Pailoor, Kyle Charbonnet, Isil Dillig, and Yu Feng. Practical security analysis of {Zero-Knowledge} proof circuits. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1471–1487, 2024.

[ZYO+24] Chengru Zhang, Xiao Yang, David Oswald, Mark Ryan, and Philipp Jovanovic. Eva: Efficient ivc-based authentication of lossy-encoded videos. *Cryptology ePrint Archive*, 2024.

A Artifact Appendix

Our artifact contains the complete proof-of-concept implementation together with the scripts required to reproduce the benchmarks reported in the paper.

A.1 Software Dependencies

The implementation requires the following software components.

- `Node.js` (LTS release).
- `snarkjs`.
- `Rust` (stable toolchain).
- `Circom`.
- Standard development tools (e.g., `gcc`, `build-essential`, `nasm`, GMP, and a JSON library).
- GNU `time` (used for benchmarking).

The artifact has been successfully evaluated on multiple systems and is not sensitive to minor dependency versions. For reference, one of our evaluation environments used:

- Ubuntu 22.04,
- Circom 2.2.1,
- snarkjs 0.7.5,
- Rust nightly 1.86.0.

A.2 Building the Artifact

After cloning the repository, the prover implementation can be compiled using:

```
cd nova
cargo build
cargo install --path .
```

The installation can be verified by executing:

```
provenview --help
```

The zkSNARK circuits can then be compiled as follows:

```
cd ../circuits
npm install
./build_circuits.sh
```

A.3 Reproducing the Benchmarks

The repository provides a benchmarking script that reproduces the experiments reported for proving vector commitment construction (Table 6).

```
./benchmark.sh
```

The script automatically executes the complete benchmarking pipeline using the sample inputs included in the repository.

A.4 Running the Prototype

The repository also includes a complete demonstration pipeline for processing arbitrary video files.

First, create the Python environment:

```
cd python
virtualenv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Execute the commitment phase:

```
python commit_phase.py
```

Execute the proving phase:

```
python prove_phase_full.py
```

When prompted, the default parameters can be accepted by pressing `Enter`.

The generated files are written to the `python/output/` directory and can subsequently be used as input to the prover:

```
provenview --function <FUNCTION>
           --resolution <RESOLUTION>
           --input <INPUT_DIRECTORY>
           --circuit <R1CS_FILE>
           --output <OUTPUT_FILE>
           --witnessgenerator <WITNESS_GENERATOR>
```