# VEIL: A System for Certifying Video Provenance

Ashish Gehani Ulf Lindqvist
SRI International
E-mail: {ashish.gehani,ulf.lindqvist}@sri.com *

## Abstract

*Traditionally, a consumer decided how much to trust a piece of data based on its source. As digital video cameras and editors become ubiquitous, an arbitrary video object is increasingly likely to be produced using a range of operations that combine clips from a multitude of sources. A consumer can determine the assurance level of the data by knowing its lineage.*

*We describe a system to embed the provenance of the video into the data itself. As long as the video contains a predefined threshold of data (from the spatial and temporal domains), the entire lineage can be ascertained. We embed the metadata using subpixel linear interpolation between similar blocks in proximal frames. It can then be extracted in real time using a novel method for computing the embedded interpolation.*

*We implemented the process in C and report the performance overhead it introduces for playing video files. We also characterize the tradeoff between the auxiliary channel's capacity (which limits the amount of provenance metadata that can be embedded) and the extent to which the video can be edited (in the spatial or temporal domains) while retaining complete lineage.*

## 1. Introduction

Digital video recordings are becoming increasingly ubiquitous. For a number of applications, the data's utility depends on how much it can be trusted. To this end, the provenance of the video plays a critical role. For example, legal prosecutors use data as forensic evidence only if its chain of custody can be established. Economic analysts weigh information according to the credibility of its origins. Military commanders take action against enemy combatants only if they know the source of intelligence data. In each case, the consumer judges the data's reliability as a function of where it originated and who manipulated it en route.

As with watermarking, a primary goal is that the changes made are imperceptible to the human visual system. While watermarks are embedded to protect the producer's rights, provenance is inserted to assist the consumer in deciding the trustworthiness of the content. As a result, the robustness of the embedding to distortions by the consumer is not of consequence. Instead, the embedded data must survive the reductions in the spatial and temporal domain that occur with legitimate editing. (For example, a watermark would not aim to guard against an operation where half the video is deleted since this would noticeably degrade the viewing experience, but provenance must survive such editing.) Further, the embedding must communicate an increasing quantity of information (representing the origins and changes made to construct the video object) while watermarks only need to convey a fixed, small quantity of information (attesting the authenticity of the data).

Each time a video object is created or modified, a provenance element capturing the operation is defined. This serves as the metadata that must be embedded into the video data. The *video embedding of information for lineage (VEIL)* certification exploits the similarity between successive video frames. Data blocks in each frame are perturbed by a subpixel quantity proportional to the metadata to be encoded. Provenance is inserted periodically throughout the data. Using the construction of Section 5.1, the metadata in a block can be "read" by calculating its subpixel displacement. Since the provenance has been redundantly inserted, when frames are cropped or segments extracted, enough metadata remains for the entire lineage to be reconstructed.

Section 2 describes the similarities and differences between digital video watermarking and the problem of embedding provenance metadata. Section 3 defines what constitutes authenticated provenance metadata for a video object. Section 4 outlines how this information can be embedded into the video data such that the goals described in Section 2 are met. Section 5 describes how the provenance can be extracted from a video file. Section 6 reports on the performance of the scheme. Section 7 describes related work on watermarking and steganography. We conclude in Section 8.

## 2. Background

The field of information hiding encodes an *embedded channel* in a *carrier medium*, such as audio, photo or video data. It has two branches, namely steganography and watermarking, with differing goals and methodologies. In the case of steganography, the carrier object is selected to prevent an adversary from detecting the existence of the em-

*This work was produced in part with support from the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

bedded channel. In contrast, watermarks are designed to be detectable even if the adversary manipulates the data. The design goals of VEIL are closer to watermarking than steganography. Here we describe them and how they differ from those of watermarking algorithms.

### 2.1. Security

The purpose of a watermark is to protect the rights of the data producer. For example, it may be used to control the number of copies made of a watermarked object [2], resolve the true ownership of a multimedia object [6], or verify that the object has not been changed from its original form [9]. In contrast, VEIL is designed to help the data consumer. If they can ascertain the lineage of the object, they can then decide how much to trust its contents. If the object's creators and modifiers are all trusted by the consumer, the estimate of its trustworthiness can be elevated. Typically, this level will be a function of the least trusted principal that provided content or made changes.

Watermarks are usually embedded in an object and then checked for by a trusted device. Since the data passes unprotected from the producer to the device in question, it can be manipulated en route. Hence the threat model for watermarking must assume an adversary that can attempt to remove the watermark while maintaining the utility of the data. Although the trusted device is under the control of the consumer, it is assumed that the verification functionality is tamper-resilient.

In the case of VEIL, an adversary has three alternatives. The first option is to certify the changes that the adversary makes. This is not an attack since the data consumer can decide not to trust the object if the adversary is not trusted. The second option is for the adversary to edit the data but leave the lineage unchanged. VEIL must guard against this. If the adversary can successfully accomplish this, then the data consumer may trust the object based on the false lineage. Because the consumer will assume that the object has not been altered by the untrusted adversary. Consequently, VEIL must ensure that such uncertified alterations can be detected. The third option for the adversary is to alter the object so that the lineage metadata is lost. This is inconvenient for the consumer but does not constitute a successful attack against VEIL. When this occurs, the consumer is forced to retrieve another copy of the object (that has not been tampered with by the adversary). Even if this is not possible, the consumer will treat the object as if its lineage is unknown and will not elevate their level of trust in its content. Thus, VEIL makes no attempt to prevent the removal of lineage metadata from an object.

### 2.2. Robustness

When operations are performed on a watermarked object, they can affect the watermark in one of three ways. The effect depends on the application for which the watermark was designed. *Fragile* watermarks [7] are destroyed by even small changes to the object. Further, their erasure is localized to the region that has been changed by the adversary. This allows a verifier to determine which part of the object has been altered without authorization. *Robust* watermarks [5] are designed to survive aggressive distortions. This property is useful for applications like copy protection. For example, a DVD duplicator can be programmed to disallow copies of watermarked videos. *Semifragile* watermarks [13] allow the attacked areas to be localized (like fragile ones) but also survive mild distortions (like robust watermarks).

VEIL decomposes the provenance into many pieces. These must all be reconstituted to reconstruct the complete lineage. Redundant copies are embedded throughout the video. If portions of the video (in either the spatial or temporal domain) are removed, there may still be enough pieces left to reconstruct the provenance. In this sense, VEIL is similar to a robust watermark. Conversely, if the distortion affects a confined region, the lineage data will be absent in a localized region as with a fragile watermark. Hence, VEIL resembles a semifragile watermark. However, its functionality does not rely on this. In particular, by removing the lineage, an adversary cannot make a consumer trust an object more than the consumer would have if it had come from an unknown producer. This holds true in any model where trustworthiness is a nonnegative quantity. (At worst, no faith is placed in untrusted objects.)

### 2.3. Visibility

When a limited amount of information (such as the copyright owner's name) is to be conveyed directly to the user, *visible* watermarks can be embedded into photo [4] or video [16] data. If the embedding is *reversible* [25], it can subsequently be removed. This is useful for limiting access to the pristine version to authorized users. Since visible watermarks have an adverse effect on the viewing experience, *invisible* watermarks are used instead when the application permits. The embedding occurs in the part of the data to which the human visual system is least sensitive. The problem is complicated because lossy compression algorithms exploit the same property. They attempt to discard the same part of the data since this yields the greatest size reduction while minimizing the impact on the viewing experience (after the object is decompressed).

A video watermark has two primary perceptual effects. The first is the noise or pattern introduced that can be discerned in a single frame. Invisible watermarks minimize this by matching the watermark's local statistical properties to those of the carrier. VEIL embeds information using bilinear interpolation. This has an effect equivalent to shifting the sampling grid by a subpixel quantity [23]. Since the gradient of the difference between the original and translated values is low, the distortion has a small upper bound. The second effect of a video watermark is "flickering" [27] caused by the insertion of different watermarking data in successive frames. If the watermark remains constant, it can be removed by averaging a sequence of frames with varied backgrounds [24]. VEIL uses differing quantities of subpixel interpolation from block to block depending on the metadata being encoded. The transformation is used in video watermarking [16] since it does not create a visual distraction.

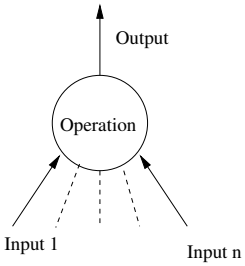


**Figure 1. A *primitive operation* transforms a set of input video files into a single output video.**

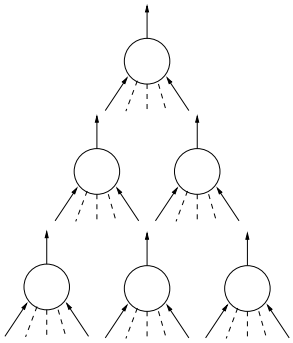

**Figure 2. A video object's provenance is a collection of primitive operations assembled into a *compound operation* tree.**

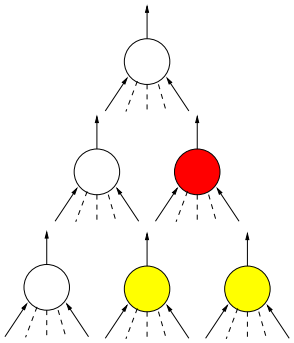

**Figure 3. If provenance is stored in an external file or metadata (such as MPEG headers), then if an operation is performed on a non-compliant node (shown in red), all metadata from preceding operations (marked in yellow) is lost.**

### 2.4. Capacity

Shannon's *capacity* bound [22] provides an estimate of how much metadata we can embed in a carrier [1]. The capacity is defined as a function of the signal and noise of the information channel. Since the effect of a video on the human visual system is not uniform, it has to be represented as a collection of information channels, each of which has its own signal and noise characteristics. Current perceptual models are not precise enough to support modeling this [14]. VEIL operates on the spatial domain. Depending on the compression algorithm used, different levels of quantization are then applied. The choice affects the noise characteristics of the channel and hence the capacity available to VEIL. Further, the maximum signal strength that can be embedded in the subpixel translation channel has not been empirically measured. (VEIL relies on the fact that the operation is perceptually tolerated on average.) Because of these three factors, VEIL's capacity cannot be precisely characterized. Analogues of each issue apply to watermarking as well. However, they are of less concern there since watermarks are often designed only to be detected (in which case a single bit is being encoded). Even when they contain a pattern, it is typically of limited, fixed length. Despite the absence of a precise measure, we still expect VEIL's capacity to be greater than that of a watermarking algorithm. This is because a watermark operates in a channel that may contain much more noise (introduced by an adversary attempting to remove the watermark). VEIL either has an error-free channel or does not provide any verifiable information.

## 3. Design

We first describe the properties with which provenance metadata should be imbued. Next, we explain our model of what constitutes lineage for a video object. We then outline how the provenance can be collected for the purpose of embedding into the video object. Finally, we detail the storage format of the metadata.

### 3.1. Desired Properties

Our goal is to use the provenance to decide how much to trust the content of the video. Therefore, the metadata should have the following properties. First, the provenance must be *authentic*. A principal must not be able to create, append or modify an element as another principal. Subsequently, other principals must be able to validate the element. Second, only operations and inputs *necessary* for reconstructing the data should be noted in the provenance. For example, if a sequence of idempotent operations is performed on the data, only one should be recorded. Third, the provenance must be *complete*. It must enumerate all the inputs used to construct the video object in question. If an object's provenance satisfies these properties, its lineage can be accurately characterized.

### 3.2. Lineage Tree

The granularity at which we track the provenance of an object affects the overhead introduced. If we attempt to trace and record the details of every operation connected to the object, the system's performance will perceptibly degrade and the metadata will grow to need more space than the data. Instead, we exploit the fact that video files cross administrative boundaries as integral objects. VEIL is not designed for streaming media. Thus, when analyzing the trustworthiness of a video object, analysis at file granularity suffices.

We define the semantics of a *primitive operation* to be an output video file, the process that generated it, and the set of input video files it read in the course of its execution. For example, if a process splices a number of segments together and outputs the result to a file, a primitive operation has been performed. The primitive operation denoted as $(Output, Executor, Input_1, \ldots, Input_n)$ is depicted in Figure 1. If a process writes out a number of video objects, a separate instance of the representation in Figure 1 is used for each output file. Primitive operations are combined into a *compound operation*, as illustrated in Figure

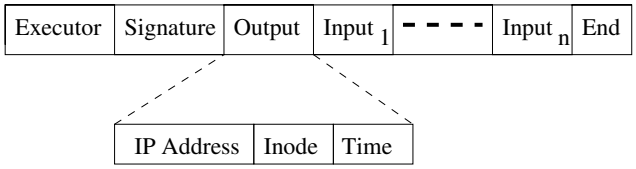


**Figure 4. Each primitive operation is stored in this format.**

| Fan-in / Levels | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 2 | 0.09 | 0.14 | 0.19 | 0.24 |
| 3 | 0.14 | 0.34 | 0.65 | 1.05 |
| 4 | 0.19 | 0.75 | 2.02 | 4.30 |
| 5 | 0.24 | 1.56 | 6.13 | 17.30 |

**Table 1. The space needed to represent a lineage tree depends on two factors. The first is the longest sequence of video reuse by different principals. This corresponds the number of levels in the tree. The second factor is the average number of video files that are used in the process of producing a single new video object. The values in the table are the number of kilobytes needed for a given number of levels and fan-in.**

2. Each vertex represents the execution of a different process. For example, if the output of splicing together several segments is then cropped to a different aspect ratio using a different application (which executes as a separate process), then combination of splicing and cropping is a compound operation. Thus every video object is the result of a compound operation that can be represented by a lineage tree. The set of all primitive operations in the tree serves as the abstract description of the lineage. We do not store the details of the process in our representation of a primitive operation. Instead we note the identity of the user who executed the process. We do this since the end user is interested in who modified the data en route rather than what specific operations were performed. This identity must have global semantics, such as a name that can be looked up using a public key infrastructure or web of trust.

At first glance, our definition of a compound operation may appear to introduce false dependencies. One may expect to be able to provide a more precise dependency set by using threads, system calls, or assembly instructions as the definition of an operation. For example, one could construct the system call control flow graph of a process and trace the possible execution paths to an output. From an information flow standpoint, only the inputs that feed into the output should be part of its dependency set. However, our goal is tracking the lineage of the final output of a compound operation. An execution sequence in a process, $P$, that does not affect the primitive operation's output but does affect the input of another process, $\hat{P}$, is called a *side effect*. An input, $I$, that produces only a side effect would not be included in the dependency set calculated based on information flow to the output. However, the output of $\hat{P}$ may then be utilized as part of the compound operation. As a result, the input $I$ would not be included in the lineage even though it should have been. This necessitates the conservative approach that we follow.

### 3.3. Storage

The producer of a video object must record the provenance metadata in persistent storage so the final consumer can access it. The information can either be stored in a metadata associated with an object or it can be embedded into the object itself, as is done with a watermark. Since the video may originate and be edited in multiple realms, filesystem attributes can not be used as they are lost when the data moves between administrative domains. Similar issues arise when the lineage is stored in an external file or in the headers of the video. This is illustrated in Figure 3. When the object is moved between systems, the accompanying file may be left behind. When video editing software manipulates the object, the metadata stored in the headers of input files are lost. Thus, we embed redundant copies of the provenance metadata throughout the data itself, in a manner that does not affect the consumer's perception of the video.

### 3.4. Metadata Format

The format of a primitive operation is depicted in Figure 4. The first element, $E$, identifies the principal that created the metadata. The information is used to look up the public key, $P_E$, of the principal. We use a 64 bit field for $E$. Since there are $2^{32}$ IPv4 addresses, this would allow an average of $2^{32}$ users per public IP address. (All the users behind a single NAT'ed IP address are associated with a single public IP address.) The second element, $S$, is the output of a digital signature that commits the executor $E$ to the output of the operation, $O$, and the set of inputs $I_1, \ldots, I_n$. We use 160 bits for the signature $S = \text{SIGN}_{K_E}(O, I_1, \ldots, I_n)$, where $K_E$ is the principal E's private signing key. This is sufficient to construct cryptographically strong digital signatures [3]. If there are no inputs (as occurs when the video object has been captured directly from a camera), then the signature's parameter is just the output file, $O$. Each input and output file is represented with a globally unique identifier. We use a 96 bit field consisting of three 32 bit parts. The first is the IP address of the host where the operation occurred. The second is the *inode* (or equivalent filesystem identifier) of the file containing the output. The third part is the time when the file was modified (using the Unix convention of counting it in seconds since January 1st, 1970). Since there are a variable number of input files, the end of the description of the operation is denoted using a special character, $\phi$, that is never used to represent metadata.

Note that $(IPAddress, Inode, Time)$ triple is used to construct a unique identifier for the state of a file. It is used to match edges during reconstruction of the provenance graph. VEIL's output is a tree of users that modified an object, not the host addresses, operation timestamps, and names of intermediate files. If the latter were required, the metadata format could be extended to incorporate full host

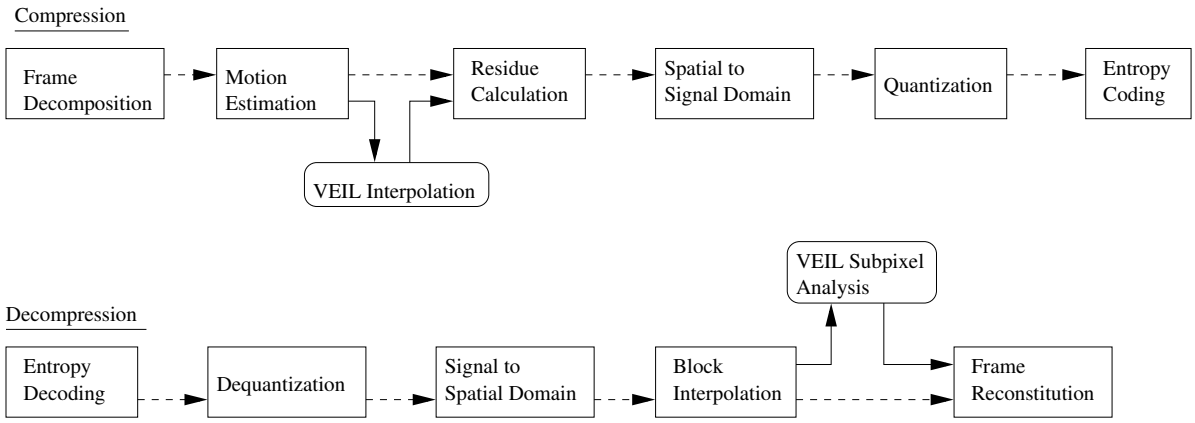


**Figure 5. VEIL intercedes after motion estimation vectors are determined during compression. The subpixel deltas are extracted after blocks have been reconstructed during decompression.**

and file names for each input and output. The inclusion of the timestamp in the identifier prevents the provenance graph from developing cycles.

Table 1 shows the amount of storage required for encoding the complete lineage of a video file. It is characterized as a function of the average fan-in at each vertex and the number of levels in the tree. The fan-in of a vertex represents the number of video files that were used as inputs when the output file from that vertex was created. Each level in the tree is created when the outputs from one principal are composed as inputs for another principal.

# 4. Embedding Lineage

We provide an overview of how VEIL embeds lineage into a video file in Section 4.1 before detailing the steps in Section 4.2 and explaining how bits are encoded using subpixel interpolation in Section 4.3.

## 4.1. Overview

Video objects consist of a sequence of frames. A small fraction are compressed using only the spatial redundancy, as occurs with image compression. (In the MPEG standard, these are the intra-coded $I$ frames.) The rest of the data is compressed by exploiting temporal correlation. Each block in a frame is replaced with a pointer to a similar one from another frame. The difference between the two blocks is encoded to allow the process to be reversed during decompression. (These are MPEG's forward predicted $P$ frames and bidirectionally predicted $B$ frames.) VEIL intercedes after the pointer has been found, but before the differences between the blocks have been computed, as illustrated in Figure 5. (If the video is not being compressed, an encoding algorithm like MPEG can be run to provide VEIL the pointers.) At this point, each block in a $P$ or $B$ frame is associated with one from an $I$ frame that it is similar to. VEIL is now in a position to encode information.

VEIL extracts the lineage metadata from each input file using the procedure described in Section 5. A new root for the lineage tree (of the form described in Section 3.4) is constructed. The new lineage tree consists of the extracted information and the new root. It is serialized into a string of bits (as detailed in Section 4.2). This is then divided into groups of bits. Each group is interpreted to be a binary number that is then normalized to provide a fraction between 0 and 1. This fraction is encoded into a block from a $P$ or $B$ frame by interpolating the block at a subpixel displacement. The original block is replaced by the interpolated version. If the video is to remain uncompressed, the procedure ends. To compress the video, the residue calculation, transformation to signal domain, quantization, and entropy encoding proceed.

## 4.2. Embedding Algorithm

A lineage tree $T = \{V_i\}$ consists of a set of elements $V_i$, each of the form described in Section 3.4. The process of embedding the lineage into the video consists of the following steps.

1. A new element $H$ is added to the set $T$. $H$ is a 160 bit element that will be used to store a signature of the video data.

2. The root of the lineage tree is prefixed with a label denoting it.

3. $\hat{T}$ refers to the modified version of $T$. Its size is calculated and denoted by $\tau$.

4. The length of the spatial domain data in the $P$ and $B$ frames of the video object is calculated. This is denoted with $\lambda$.

5. If the video data is to be compressed, the block size of the standard is denoted by $\beta$. If the data is to remain uncompressed, $\beta$ can be selected freely. If MPEG compression is used, a typical value for $\beta$ is 64 (using 8x8 blocks of pixels).

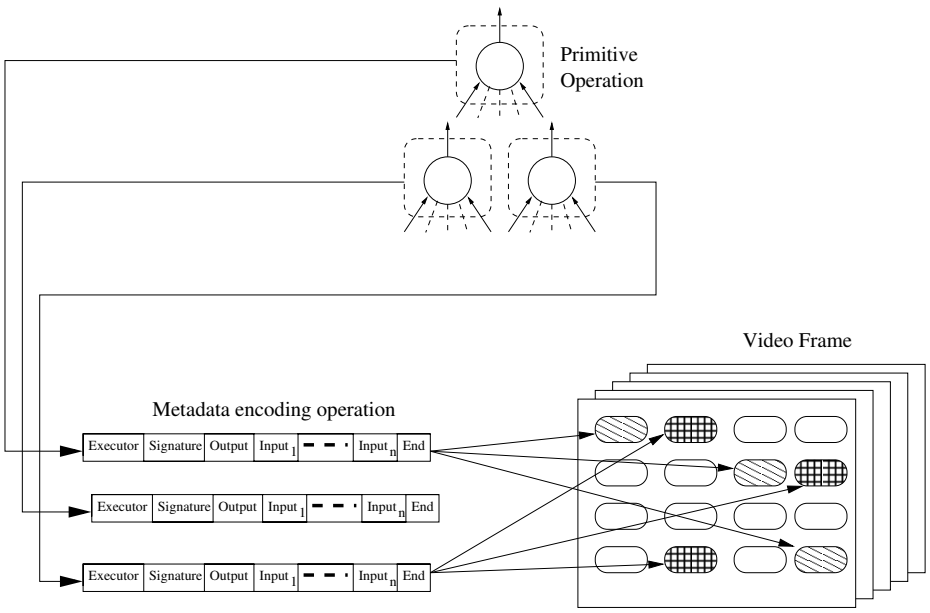


**Figure 6. Each primitive operation is represented by a metadata element (of the format shown in Figure 4). The lineage consists of a set of such elements, each of which is then redundantly encoded in the frames of the video.**

6. VEIL's capacity is parameterized. The parameter that controls this is $\alpha$ and must be decided prior to embedding.

   As $\alpha$ increases, the amount of information that VEIL will attempt to embed in a fixed length of video will increase exponentially. However, the "lossy" operations of transforming to the signal domain and quantization occur after the metadata is embedded. This means that as $\alpha$ increases, there is a higher probability that the encoded metadata will be destroyed by the "lossy" operations. Thus, the smallest value of $\alpha$ that provides sufficient capacity is selected. This is done as follows.

   (a) A reliability factor $\rho$ is selected. This represents the number of times the lineage will be redundantly inserted into the video object.

   (b) $\alpha = \lceil \frac{\tau.\rho.\beta}{\lambda} \rceil$ since $\tau.\rho$ bits must be encoded in $\frac{\lambda}{\beta}$ blocks, each with an encoding capacity of $\alpha$ bits.

7. $\rho$ copies of $\hat{T}$ are made. Each one is independently randomly permuted. These are then concatenated into a single string, $\Sigma$. This is the redundant representation of the lineage tree that will be embedded into the video object, as illustrated in Figure 6. The random permutation decreases the probability that all $\rho$ copies of any element will be unreadable after "lossy" compression.

8. $\Sigma$ is divided into groups of $\alpha$ bits. Each is treated as a number in the range from 0 to $2^\alpha - 1$. This is normalized (with a denominator of $2^\alpha$) to obtain a value, $\delta x$, in the range of 0 to 1. $\Sigma$ is thus converted into a stream of $\delta x$ values.

9. The $i^{th}$ $\delta x$ value is encoded by interpolating the $i^{th}$ block (of all the $P$ and $B$ frames). Interpolation is detailed in Section 4.3.

10. The last $\alpha$ blocks are encoded with $\delta x = \frac{1}{2^\alpha}$. The decoder needs this value to know how much to scale the values to invert the normalization.

11. A signature is computed over all the data in the modified video object except for the blocks where $H$ is to be encoded. Unlike $S$ in Section 3.4 (which is computed over object identifiers), $H$ is computed over the object's content. The blocks where $H$ is to be encoded are interpolated to embed the signature's bits. The signature binds the lineage tree to the specific video object.

    The security of $H$ is guaranteed through circularity. The blocks in which the signature is encoded are not part of the input to the signature. If they change, the output of the signature will not be altered. However, this will cause *the recorded output of the signature to change*. On the other hand, if the blocks used to encode $H$ are changed, *the verification will fail.*

    The actual signature scheme used for $H$ depends on the level of security required. Geometric hashes [28] provide the weakest attestation while tolerating the most types of editing operations. Cryptographic hashes are the most sensitive to changes in the object, providing the strongest integrity assurance but no tolerance for video editing operations. Perceptual hashes [18] strike a balance, yielding the same hash after acceptable transformations.

### 4.3. Interpolation

We use subpixel linear interpolation to encode information. The operation will take a reference block, a contextual block and a subpixel displacement, $\delta x$, as its input. Our goal is to construct a replacement for the reference block with the following property. When the resulting block is displaced by the input subpixel quantity, the difference between the contextual block and it is minimized. Visually, this occurs when the objects in the two blocks "line up" as illustrated in Figure 7.

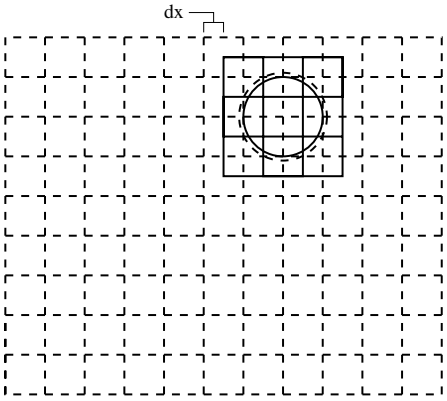


**Figure 7. Subpixel alignment of two blocks minimizes the visual distance between them.**

Linear interpolation of video data is widely used (for increasing frame resolution or embedding watermarks, for example). However, our goal is not to interpolate the input block by $\delta x$. Instead it is to produce an interpolated block that is $\delta x$ away from the contextual block. To see why this is different, consider what occurs when a camera records video. As depicted in Figure 8, different sets of (neighboring) points are registered from frame to frame. This introduces subpixel differences in blocks that are found using integral pixel motion estimation. We denote the inherent subpixel difference by $\delta i$. Therefore our goal is to translate

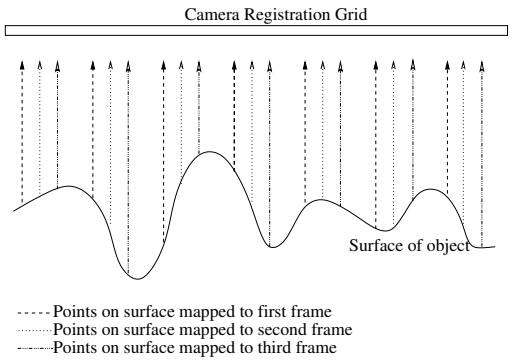


**Figure 8. A camera sensor registers slightly different locations on an object from frame to frame.**

the reference block by $\delta x - \delta i$. When the resulting block and the contextual block are compared, they will appear to be at a distance of $\delta x$ from each other. $\delta i$ is determined using our algorithm described in Section 5. After this, every pixel in the reference block is replaced with an interpolated value. For example, if the pixel's value is $v_1$ and the value of the pixel to the right is $v_2$, then the new value is $v_1 + (\delta x - \delta i)(v_2 - v_1)$. Except for pixels at the edge of the frame, a neighbor will always exist. If the pixel is at the edge of the block, the neighbor can be retrieved from the edge of the adjacent block.

## 5. Extracting Lineage

**Algorithm 5.1:** CHECKLINEAGE($D$)

$\{E, S, O, I_1, \ldots, I_n\} \leftarrow$ GETROOT($D$)
OUTPUT($E$)
$P_E \leftarrow$ PKILOOKUP($E$)
**if** $I_1, \ldots, I_n = \{\}$
  **then** { $Result \leftarrow$ VERIFY($P_E, S, O$); **if** $Result =$ FALSE **then** CheckFailed
  **else** { $Result \leftarrow$ VERIFY($P_E, S, O|I_1| \ldots |I_n$); **if** $Result =$ TRUE **then** { **for** $i \leftarrow 1$ **to** $n$ **do** CHECKLINEAGE($I_i$) } **else** CheckFailed }

After a video object has been decompressed (if it was in a compressed format), each block in a $P$ or $B$ frame is associated with a contextual block. If the video was not compressed, a compression algorithm's motion estimation phase can provide pointers to these blocks. Given such a pair of blocks, we wish to calculate the subpixel displacement $\delta x$ that minimizes their visual distance. The value will provide $\alpha$ bits of the encoded lineage. One method of computing this is to select a small value $\delta b$, and then try multiples of $\delta b$ to see which results in the best match. This is how block motion is estimated at integral pixel granularity by video compression algorithms. If $\alpha$ was known in advance (which it is not), then all $2^\alpha$ values in the range of 0 to $2^\alpha - 1$ can be tried. However, this is computationally prohibitive. In fact, the cost is so high that random subpixel displacement is used to secure video watermarks [16].

We have developed an algorithm that exploits the fact that values were embedded using linear interpolation. It calculates $\delta x$ using only $\mathcal{O}(\beta)$ space and time, where $\beta$ is the block size. It is equivalent to trying just one of the possible values of $\delta b$, rather than needing $2^\alpha$ or more trials. The details are presented in Section 5.1. This allows us to extract a value $\delta x$ from each block in each $P$ or $B$ frame (or its uncompressed analogue if the video was not compressed).

Each $\delta x$ is converted to a value in the range of 0 to $2^\alpha - 1$ by multiplying it by a scaling factor. This factor is determined as follows. $\delta x$ for the last few blocks is computed. The value should be the same for each block since $\frac{1}{2^\alpha}$ was encoded in the last $\alpha$ blocks. The scaling factor is calculated

as $\frac{1}{\delta x} = 2^{\alpha}$. With this, the stream of $\delta x$ values is converted to a stream of bits. Every group between two $\phi$ values constitutes a vertex in the lineage tree (or equivalently a single primitive operation).

Construction of the lineage tree requires only a single copy of each vertex. (Recall that there are $\rho$ redundant copies.) Thus, even if the video has been cropped or segments have been removed, enough copies of each vertex may remain. However, verification requires the entire video object. This is because the digital signature $H$ is computed over the entire video object (except for the blocks in which the signature's bits are encoded). $H$ is verified by computing over the video object as described in Step 11 of Section 4.2. Instead of the signing key, the verification key of the owner, $P_E$, is used. If the verification fails, the object has been tampered with en route and the lineage tree computed may belong to a different video object than the one in which it is embedded.

The procedure for recovering the lineage tree can operate independently of the verification of $H$. It is done by recursively checking each provenance element, shown below as Algorithm 5.1. The GETROOT function finds the vertex whose output matches its input parameter, $D$. It outputs the principal $E$ who executed the operation. As the CHECKLINEAGE function recurses, the output from this step enumerates all the principals that have modified the data utilized in producing the video object. The principal's signature validating the set of inputs used to produce the output is verified (even if there are no inputs) with the VERIFY function. Then the lineage of each input (if any exist) is recursively checked. If at any point the signatures fail, the function halts and notifies the user.

### 5.1. Calculating Subpixel Displacements

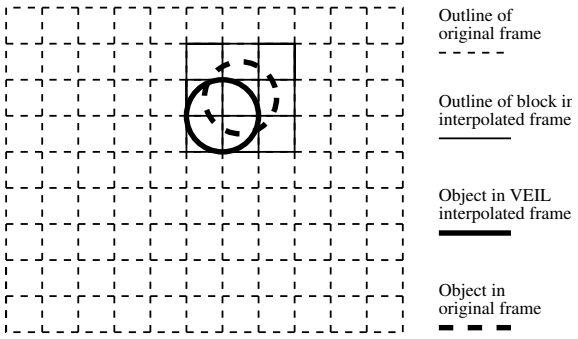


**Figure 9. After a block has been interpolated by $\delta x$, it appears displaced by a subpixel quantity. This distortion is tolerated by the human visual system.**

The lineage metadata was embedded into video frames by interpolating blocks at subpixel distances. Figure 9 depicts this. To extract the encoded information, we need to determine those subpixel quantities. This section describes how the values can be computed efficiently. We assume the pixel intensity values of the two blocks are represented by the functions $\mathcal{F}(x, y)$ and $\mathcal{G}(x, y)$ whose inputs range from $(x_{min}, y_{min})$ to $(x_{max}, y_{max})$

**Theorem 1** *$\delta x$ can be calculated in $\mathcal{O}(\beta)$ time using $\mathcal{O}(\log \beta)$ space.*

**Proof**

We assume the optimal subpixel displacement is $\delta x$. Thus if we use Mean Squared Error as our matching criterion, we obtain the equation:

$$MSE(\delta x) = \frac{1}{\beta} \sum_x \sum_y \left[\mathcal{F}(x + \delta x, y) - \mathcal{G}(x, y)\right]^2 \quad (1)$$

Assuming that the optimal subpixel displacement along the $x$ axis of the contextual block under consideration is the $\delta x$ that minimizes the function $MSE(\delta x)$ in Equation 1, we will solve for $\delta x$ using the constraint:

$$\frac{d}{d(\delta x)} MSE(\delta x) = 0 \quad (2)$$

$$= \sum_x \sum_y \left[ \left[\mathcal{F}(x + \delta x, y) - \mathcal{G}(x, y)\right] \frac{d}{d(\delta x)} \mathcal{F}(x + \delta x, y) \right]$$

Since $\mathcal{F}(x, y)$ is a discrete function, we use linear interpolation to approximate it as a continuous function for the purpose of representing $\mathcal{F}(x + \delta x, y)$ and computing $\frac{d}{d(\delta x)} \mathcal{F}(x + \delta x, y)$. We can do this since we embedded the metadata using linear interpolation.

$$\forall \delta x,\ 0 \le \delta x \le 1 :$$
$$\mathcal{F}(x + \delta x, y) = \mathcal{F}(x, y) \ + \ \delta x \left[\mathcal{F}(x + 1, y) - \mathcal{F}(x, y)\right] \quad (3)$$

$$\frac{d}{d(\delta x)} \mathcal{F}(x + \delta x, y) = \mathcal{F}(x + 1, y) - \mathcal{F}(x, y) \quad (4)$$

Applying Equations 3 and 4 to Equation 2, and grouping to separate terms with $\delta x$, we obtain:

$$\begin{aligned} 0 = \sum_x \sum_y [\ & \delta x \left[\mathcal{F}(x + 1, y) - \mathcal{F}(x, y)\right]^2 \\ & + \mathcal{F}(x + 1, y)\mathcal{F}(x, y) \ - \ \mathcal{F}^2(x, y) \\ & + \mathcal{F}(x, y)\mathcal{G}(x, y) \ - \ \mathcal{F}(x + 1, y)\mathcal{G}(x, y) \ ] \end{aligned}$$

By rearranging terms, we can obtain the closed form solution:

$$\delta x = \frac{\sum_x \sum_y \ [\ \left[\mathcal{F}(x + 1, y) - \mathcal{F}(x, y)\right] \left[\mathcal{F}(x, y) - \mathcal{G}(x, y)\right] \ ]}{\sum_x \sum_y \ \left[\mathcal{F}(x + 1, y) - \mathcal{F}(x, y)\right]^2}$$

The closed form solution for each $\delta x$ adds $\beta$ terms in both the numerator and the denominator. Each term requires two subtractions and one multiplication. This is followed by computing the final quotient of the numerator and denominator summations. The time complexity is therefore $\mathcal{O}(\beta)$. Since only the running sum of the numerator and denominator need to be stored, the space needed is $\mathcal{O}(\log \beta)$. □

## 6. Evaluation

Linear interpolation is widely used in photo and video editing. Efficient implementations exist that can be used for encoding the lineage metadata. However, recovering this information requires calculating subpixel displacements between blocks. We described an efficient algorithm for this in Section 5.1. We implemented the primitive in C. The timing was performed on a 2 GHz Intel Core Duo processor running Mac OS 10.4.8. As can be seen from Table 2, the provenance data can be extracted from the video in real time. For example, using a block size of 8x8, it takes about 0.7 seconds to extract all the provenance encoded in 1 second of video data. In practice, a single copy of the lineage tree will likely be recovered in seconds. After this, the overhead will drop to 0 since no more metadata needs to be extracted.

| Block size | Time to compute $\delta x$ (in $\mu$s) | Overhead (in sec) for 1 sec of video |
|---|---|---|
| 4x4 | 1.3 | 0.723 |
| 8x8 | 5.2 | 0.746 |
| 16x16 | 18.2 | 0.648 |
| 32x32 | 72.2 | 0.649 |
| 64x64 | 292.1 | 0.657 |

**Table 2. The time to compute $\delta x$ is linearly dependent on the block size, as Section 5.1 predicts. The test video has a resolution of $640$x$480$ and 30 frames per second.**

VEIL embeds redundant copies of the lineage tree into the video. The length of video needed to recover the lineage tree depends on a number of factors. These include the number of levels and the average fan-in of the provenance tree, the block size, the number of bits encoded per block, and the number of copies of the tree that were embedded. In Table 3, we characterize the video length that suffices to recover a lineage tree. The tree in question has an average fan-in of 4 and 4 levels. It needs $35,264$ bits of storage. 5% of the video consists of $I$ frames [11].

As can be seen from Table 3, even if a conservative 2 bits are encoded per block (that is, $\delta x$ can take on $2^2 = 4$ values), only 0.13 seconds of video are required to reconstruct the entire lineage tree. We include the last line in the table (where 6 bits are encoded per block) for comparison purposes. Since this would require $2^6 = 64$ levels, the subpixel computation would reach its reliability threshold if "lossy" compression introduced errors much greater than 1%. Further empirical analysis is required to determine the maximum value $\alpha$ can take on. However, even with low $\alpha$ values, VEIL provides sufficient capacity to embed lineage trees robustly.

## 7. Related Work

We now describe the relationship between our work and earlier research. We point out the properties required for our application that were not satisfied by previous approaches.

| Bits encoded per block ($\alpha$) | Block size in bits ($\beta$) | Redundant copies in 1 min of video ($\rho$) | Length for 1 lineage tree (in sec) |
|---|---|---|---|
| 2 | 8x8 | 465 | 0.13 |
| 4 | 8x8 | 931 | 0.06 |
| 4 | 16x16 | 232 | 0.26 |
| 6 | 8x8 | 1396 | 0.04 |

**Table 3. VEIL's metadata embedding capacity is characterized as a function of $\alpha$, $\beta$ and $\rho$.**

### 7.1. Provenance

Data provenance has a range of applications. Numerous systems have been built to track it. Compaq SRC's Vesta [10] uses it to make software builds incremental and repeatable. Trio [26] models it to track tuples in a relational database. Lineage File System [21] records the input files, command line options, and output files when a program is executed. Its records are stored in a SQL database that can be queried to reconstruct the lineage of a file. Provenance-Aware Storage System [20] augments this with details of the software and hardware environment. In each of these systems, provenance metadata is stored separately from the data objects. Thus it can be used only on the host where the system is running. If the data is transferred to a remote location, the provenance is lost. VEIL is specifically designed to embed the metadata into the data so that it is transparently transferred along with the object when it crosses from one system to another.

### 7.2. Steganography

Steganography introduces a covert channel into a carrier medium (such as images or video) to transmit information. The channel can be created in a variety of ways. For example, the least significant bits in the spatial domain [8] of an image can be used to embed information (since the human eye is least sensitive to changes in the highest frequencies. Alternatively, the insertion could occur directly in the frequency domain [15]. Because video objects have high capacity, the embedding can occur in the most amenable regions, such as those that appear noisy [17]. Steganography selects carrier data that will minimize the chance of the embedded information being detected. However, we need to add lineage to every file. VEIL is designed to be agnostic to the properties of the carrier medium and can operate on arbitrary video data. Further, the provenance metadata is not secret. Therefore VEIL can utilize capacity in the underlying medium that is not available to steganography algorithms.

### 7.3. Watermarking

Watermarking is the other branch of information hiding. It allows metadata to be embedded in the data using encoding methods similar to those of steganography. Since the goal is to prevent an adversary from removing the embedded metadata (unless the adversary severely distorts the carrier data as well), the encoding is determined by the attacks that are being guarded against. For example, image

watermarks aim to survive rotation, translation and scaling [12] or JPEG compression and cropping [19], video watermarks protect against frame averaging [24], and more general multimedia watermarks use spread-spectrum encoding to persist through intermediate conversions to analog format, requantization, or resampling [5]. A significant difference from steganography is that the carrier medium (such as the image whose copyright is being protected) for watermarking is not selected by the embedding algorithm. Therefore, in principle, lineage could be inserted in a file using a watermark. However, watermarking algorithms assume that the carrier data's overall substance cannot be significantly altered since it would no longer be acceptable to the consumer. In contrast, VEIL must provide lineage information even if the operation performed makes a large change, such as cropping out the left half of each frame or retaining only the second half (in time) of the video clip. Further, watermarks use a fixed capacity while VEIL must address the need for increasing capacity since lineage grows each time the data is altered.

## 8. Conclusion

We have described VEIL, a method to embed the lineage of a video clip into the data itself. Using VEIL, a consumer can determine the set of users that created and modified all the data that is part of the video object. This will allow consumers to make an informed choice regarding how much to trust the video.

Since VEIL embeds the metadata directly into the video data using subpixel interpolation, it imposes no storage overhead, allows the video to be played by decoders without VEIL functionality, and requires no auxiliary lineage files to be managed.

VEIL extraction operates fast enough to be used for real-time recovery of lineage. For example, it takes $0.06$ seconds to extract the complete lineage tree generated by $4$ generations of video modifications made by upto $21$ principals, each using an average of $4$ input files. (The example assumes that $4$ bits are encoded in each block of size $8$x$8$.) This property is due to VEIL's algorithm for efficiently calculating subpixel matches.

## References


[1] M. Barni, F. Bartolini, A. De Rosa and A. Piva, Capacity of the watermark channel: how many bits can be hidden within a digital image, Proceedings of SPIE, Vol. 3657, 1999.

[2] J. Bloom, I.J. Cox, T. Kalker, J.-P. Linnartz, M.L. Miller and C.B.S. Traw, Copy protection for DVD video, Special issue on identification and protection of multimedia information, Proceedings of the IEEE, Vol, 87(7), 1999.

[3] Dan Boneh, Ben Lynn and Hovav Shacham, Short signatures from the Weil pairing, Proceedings of Asiacrypt, Lecture Notes in Computer Science, Vol. 2248, 2001.

[4] G. W. Braudaway, K. A. Margerlein and F. C. Mintzer, Protecting public-available images with a visible image watermark, Conference on Optical Security and Counterfeit Deterrence Techniques, Proceedings of SPIE, Vol. 2659, 1996.

[5] I. Cox, J. Kilian, T. Leighton and T. Shamoon, Secure spread spectrum watermarking for multimedia, IEEE Transactions on Image Processing, Vol. 6(12), 1997.

[6] S. Craver, N. Memon, B. L. Yeo and M.M. Yeung, Resolving rightful ownership with invisible watermarking techniques: limitations, attacks and implications, IEEE Journal Selected Areas Communication 4 (16), 1998.

[7] J. Fridrich, Image watermarking for tamper detection, Proceedings of the IEEE International Conference on Image Processing, Vol. 2, 1998.

[8] J. Fridrich, M. Goljan and R. Du, Reliable Detection of LSB Steganography in Color and Grayscale Images, Proceedings of the ACM Workshop on Multimedia and Security, 2001.

[9] J. Fridrich, Security of fragile authentication watermarks with localization, Security and Watermarking of Multimedia Contents IV, Proceedings of SPIE, Vol. 4675, 2002.

[10] A. Heydon, R. Levin, T. Mann and Y. Yu, The Vesta Approach to Software Configuration Management, Technical Report 168, Compaq Systems Research Center, 2001.

[11] M. Krunz, R. Sass and H. Hughes, Statistical characteristics and multiplexing of MPEG streams, Proceedings of the 14th IEEE INFOCOM, Vol. 2, 1995.

[12] M. Kutter, Watermarking resisting to translation, rotation, and scaling, Proceedings of SPIE Multimedia Systems and Applications, Vol. 3528, 1998.

[13] E. Lin, C. Podilchuk and E. Delp, Detection of image alterations using semi-fragile watermarks, Security and Watermarking of Multimedia Contents II, Proceedings of SPIE, Vol. 3971, 2000.

[14] C. Y. Lin and S. F. Chang, Zero-Error Information Hiding Capacity of Digital Images, IEEE International Conference on Image Processing, 2001.

[15] L. Marvel, C. Boncelet, Jr. and C. Retter, Spread-spectrum image steganography, IEEE Transactions on Image Processing, 1999.

[16] J. Meng and S. F. Chang, Embedding visible video watermarks in the compressed domain, Proceedings of the IEEE International Conference on Image Processing, Vol. 1, 1998.

[17] Hideki Noda, Tomofumi Furuta, Michiharu Niimi and Eiji Kawaguchi, Video steganography based on bit-plane decomposition of wavelet-transformed video, Security, Steganography, and Watermarking of Multimedia Contents VI, Proceedings of SPIE, Vol. 5306, 2004.

[18] J. Oostveen, T. Kalker and J. Haitsma, Visual Hashing of Digital Video: applications and techniques, SPIE Applications of Digital Image Processing, 2001.

[19] C. Podilchuk and Z. Wenjun, Image-adaptive watermarking using visual models, IEEE Journal on Selected Areas in Communications, Vol. 16(4), 1998.

[20] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun and Margo Seltzer, Provenance-Aware Storage Systems, Proceedings of the USENIX Annual Technical Conference, 2006.

[21] http://crypto.stanford.edu/ cao/lineage.html

[22] C. E. Shannon, A mathematical theory of communication, Bell System Technical Journal, Vol. 27, 1948.

[23] K. Su, D. Kundur and D. Hatzinakos, A Content-Dependent Spatially Localized Video Watermarked for Resistance to Collusion and Interpolation Attacks, Proceedings of the IEEE International Conference on Image Processing, 2001.

[24] K. Su, D. Kundur and D. Hatzinakos, A Novel Approach to Collusion-Resistant Video Watermarking, Security and Watermarking of Multimedia Contents IV, Proceedings of SPIE, Vol. 4675, 2002.

[25] A. van Leest, M. van der Veen and F. Bruekers, Reversible image watermarking, Proceedings of IEEE International Conference on Image Processing, Vol. 2, 2003.

[26] J. Widom, Trio: A System for Integrated Management of Data, Accuracy and Lineage, Conference on Innovative Data Systems Research, 2005.

[27] Stefan Winkler, Elisa Drelie Gelasca and Touradj Ebrahimi, Toward perceptual metrics for video watermark evaluation, Applications of Digital Image Processing, Proceedings of SPIE, Vol. 5203, 2003.

[28] H. J. Wolfson and I. Rigoutsos, Geometric hashing: an overview, IEEE Computational Science and Engineering, Vol. 4(4), 1997.