The Truth, the Whole Truth, and Nothing but the Truth – Video Attestation for Evidentiary Purposes

Malcolm Weir†
†Ampex Data Systems Corporation, Hayward, California USA,
mweir@ampex.com

Abstract:
Explores cryptographic techniques for providing assurance that recorded video or other sequential data has not been manipulated or otherwise tampered with, particularly with regard to AI tampering and systems relying on AI for real-time monitoring.

Key words: *Attestation*, e*ncryption, metadata*

Introduction

Technological developments in recent years have combined to exacerbate a long-standing issue: regardless of the literal truth of the old saying that "the camera cannot lie", people can, and frequently do, manipulate and distort what the camera captured.

So the challenge from an evidentiary perspective is knowing how much credence to give a particular photo or video. While there's nothing much that technology can do about purely optical issues (e.g. the camera pointing the wrong way), there are ways to detect efforts to edit or manipulate the evidence.

This paper focuses on military applications more than civilian ones, but many of the principles and solutions discussed are applicable to both.

Background

The first relevant development was the shift to digital video, which has facilitated various forms of manipulation that had previously been technically more challenging and required specialist equipment. In addition, analog imagery, by its nature, can support a much broader range of detail and color, which makes it harder to produce undetectable forged analog video (digital has a finite number of shades of grey, while analog can acquire a literally infinite number). The parallel with still images is that, while it was possible for a skilled artist to manipulate a photograph in a darkroom, it was had to do it perfectly, as the photos below show.

The second development relates to the mobility of cameras and duration of potential video evidence. While static CCTV has been capturing fixed viewpoints 24x7 for decades, that material contained its own reference baseline: one *knows* what the street scene should look like, because

The darkroom technician has partially concealed the manipulation by putting a fake blemish over the "new" part of Stalin's coat (left elbow).

one typically has hours of material with which to compare the potentially manipulated footage. An analyst can examine things like shadows (or the lack thereof) and make conclusions based on that.

But with mobile cameras, the viewpoint is constantly changing, and there's no reference with which to compare suspect material: a seascape is entirely dynamic and therefore virtually impossible to conclude that the pattern of light on the water is forged and concealing something more interesting!

The traditional approach to validating the authenticity of video evidence is to have a human being corroborate the recording. This is reasonable when manned platforms (like the P-8 Poseidon maritime patrol aircraft in service with the militaries of the USA, IND, KOR, DEU, GBR, AUS, NOR, NZL plus SGP and CAN soon, and the venerable P-3 Orion, which is mostly retired now).

Even with unmanned vehicles, there was generally an operator monitoring the video feed from the platform. But now that's changing: long-duration autonomous vehicles don't, by definition, need constant supervision, particularly while transiting to and from a patrol area.

A further complication is that the emerging sophistication of so-called "AI" (or "Machine Learning" and other terms) facilitates both forgery and truly unattended operation: if one trusts the "AI" to alert one to "interesting things" (however defined), then obviously there will be less pressure for continual monitoring.

The rest of this paper considers various approaches that may be used, separately or in conjunction, to create confidence that what you see on the video was what the camera saw.

Storage Encryption

Probably the most straightforward approach to protecting acquired video is to encrypt the storage / filesystem used to store the video files.

While in many applications using this approach is necessary, it is generally not sufficient. Specifically, encrypting the storage serves many goals related to confidentiality, but it is not as useful for demonstrating authenticity: anyone with the encryption key has full access to do whatever they wish with the video files. And by encrypting the entire storage volume, there's no granularity: every video file is accessible to anyone with the key.

A variant of this approach is to use file-based encryption, or video-stream encryption (which is, to all intents and purposes, much the same thing): instead of encrypting the entire storage area, only the video data or files are encrypted. This allows for granularity but adds significant complexity to the key handling challenge: each video stream / file needs a unique key. It is also worth noting that file metadata might reveal sensitive information through metadata analysis: an interested adversarial party may not be able to see what the video shows, but by looking at file timestamps they might discover information about when it was collected.

The "unique key" challenge can be addressed by using a scheme like Ampex's Turnstile, where each file or video segment is encrypted with a random "data encrypting key" (DEK), which is in turn encrypted and saved using a public/private algorithm where the recording platform only has the public half of the key. This does, however, mean that the recording platform cannot decrypt previously recorded data, as (by design) it lacks the private decryption key.

But no matter how it's implemented, encrypting the video file only provides benefits while the key is protected.

However, for evidentiary purposes, this sort of encryption sometimes can be adequate, if not elegant; it is in essence not a lot different from an "evidence locker" in which physical evidence may be stored pending a trial. But the critical point is whether the custodian is considered trustworthy or reliable; it's generally the case in criminal prosecutions that the evidence custodian *is* also the prosecutor, leading to there being a perception of partiality and thus a lack of confidence that any video data provided is unmodified from when it was shot.

Hash Operations

The objectives described in this paper are primarily to prevent or detect manipulation of evidence. While the encryption schemes discussed above certainly can achieve those goals, they impose a heavy cost: the video cannot be viewed unless the encryption is unlocked!

But to detect changes to digital data, there is a group of algorithms known as hash functions or message digest algorithms. These have the property of taking an arbitrarily large input and outputting a smaller, fixed-size output, typically of the order of a few bytes (or tens of bytes) long.

A good hash function has the property that a small change in the input will result in a significant change in the output; this property tends to imply an even distribution of outputs across the range of possible values. An additional property of "cryptographically useful" hash functions is that it should be computationally hard to reverse the function: given an output, figure out the set of inputs that would create that output.

This property, sometimes known as "one-way functions", is essential in modern cryptosystems: the authenticity of software (etc) can be verified by examining the hash function output, as long as it's not feasible to figure out how to something that hashes the same, but is functionally quite different. Indeed, several formerly widely used algorithms (including "MD5" and "SHA-1") were retired when methods to reverse them was discovered.

For video attestation, while any hash algorithm can be used to validate the integrity of a video file in a benign environment (i.e. one in which no one is trying to produce a forgery), a cryptographic hash can be used to detect even malicious integrity, because the forger can't fabricate a plausible video that produces the same hash as the authentic original. Specifically, as cryptographic hash functions are hard to reverse, it is computationally hard to start with a hash and produce an input that yields that specific hash, let alone one that matches the format constraints of a video file!

For modern systems, SHA-384 produces a 384-bit (48 byte) output and is believed to be secure

(cryptographically speaking) even with the development of what's referred to as "cryptographically relevant quantum computers". The 384 bit value can be represented as a 64 symbol "Base64" string, using upper and lower case Roman/Latin letters, 10 Arabic numerals, and the symbols "-" and "_" (or sometimes "+" and "/", but the former is preferred). While a 64-character Base64 string is a little unwieldy, it's an improvement over the 96 hexadecimal characters that would be required!

Thus, one approach to detecting manipulation of video is simply to distribute an appropriate hash value with the video. However, providing a strong link between the hash and the source file is a challenge: keeping them in the same encrypted storage would work, but that's subject to all the same problems as mentioned above.

Signing the Hash

The issue of a wrongdoer manipulating a video's hash value can be solved using public key cryptography.

Public key cryptography, also known as asymmetric cryptography, is a method of securing information using a pair of mathematically related keys: a public key and a private key. The public key can be shared openly and is used to encrypt data or verify digital signatures, while the private key is kept secret and is used to decrypt data or create signatures. This system allows secure communication without the need to exchange a secret key in advance, solving a fundamental problem in cryptography. It is widely used in technologies such as HTTPS, email encryption, and digital certificates, often in combination with faster symmetric encryption methods for handling large amounts of data.

For this type of application, an algorithm called Elliptical Curve Digital Signature Algorithm (ECDSA) is very well suited, due to its key sizes and performance. The algorithm uses geometric mathematics on specially chosen curves to create secure key pairs, such that given on half of a pair you can only reverse the operation using the other half. Related to the ECDSA algorithm are the encoding rules (DER) that allow the signature to be represented in a self-describing structure, so that validating it can be performed by any software that understands the format.

The use of public key cryptography adds an additional useful property to the process: authentication. The principle here is that since the private key is never shared outside of the creator's environment (e.g. the video recording system), only that environment could have created the signed hash of the video, and therefore the video not only hasn't been modified but also can only have come from the specified source (assuming typical "key hygiene" where the private key is properly protected).

Post-Quantum Considerations

While the initial effort supporting this paper took place in an environment where a Quantum-Relevant Computer was just a concept on the far horizon, the reality is that US government guidance has a goal of the "old" asymmetric algorithms being mostly retired by the year 2030 and being entirely replaced by 2033.

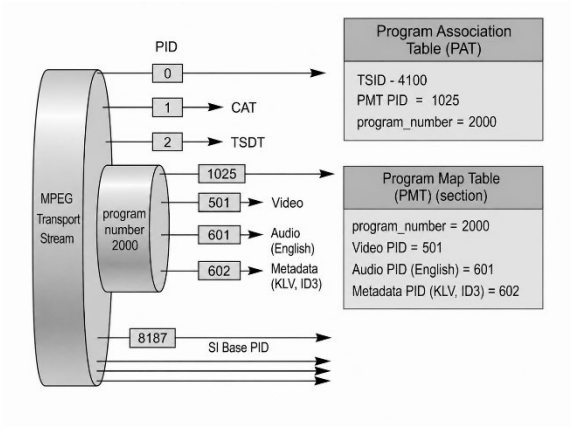
The closest successor to ECC is the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM), which is sometimes referred to by its older name of Kyber, and which is standardized by FIPS 203. As part of the post-quantum algorithm update, there are two other options that may be relevant to this application: the Module-Lattice-Based Digital Signature Standard (ML-DSA) codified by FIPS 204 and the Stateless Hash-Based Digital Signature Algorithm (SLH-DSA), in FIPS 205. Which of the three is most suitable to a given application is beyond the scope of this paper, but involves considerations such as the performance cost of the size of the result: while the result sizes are all larger than those of the legacy algorithms, the performance issue is inverted in some cases, with the new algorithms being as much as 6 or 7 times faster than the old.

Video Stream Structure

While it's common to think of and treat a video stream as a single entity, in practice "video" has typically referred to both the actual imagery and the accompanying audio channel(s). For instance, legacy standard-definition broadcast television formats like NTSC, PAL, and SECAM transmit the audio signal separately: NTSC uses an audio carrier frequency 4MHz above the video carrier frequency. Movie films have been grafting separate audio onto celluloid for a century!

With digital video, though, the video and audio signals (if any) are transmitted as packets in a single stream. For example, the MPEG Transport Stream (sometimes referred to as the MPEG-2 Transport Stream due to the timing of its standardization) allows for multiple "programs" each with multiple audio, video and metadata streams (collectively, "elementary streams") to be transmitted concurrently – and obviously useful capability for a TV broadcaster!

(Note that there are other video transports apart from the MPEG Transport Stream, but in general and by far the most common is the Transport



MPEG Transport Stream Structure

Stream, and alternatives provide essentially the same functionality using different techniques).

It's worth noting that the stream structure is independent from the encoding technique, so the video elementary stream can be encoded as MPEG-4, H.265 or even in the low-latency JPEG-XS format, while the audio can use MP3, AAC or uncompressed WAV. The metadata elementary stream also has variants, but "Key-Length-Value" (KLV) is by far the most versatile.

The Transport Stream's definition allows for, by design, an enormous number of legal variations. Layered standards such as STANAG 4609 restrict the allowable options somewhat, but for our purposes these added restrictions don't have much, if any, impact.

Embedding Signed Hashes

It is an obvious step to embed a signed hash ("integrity data") into the MPEG-2 Transport Stream.

There are broadly three design decisions to be made when implementing this approach.

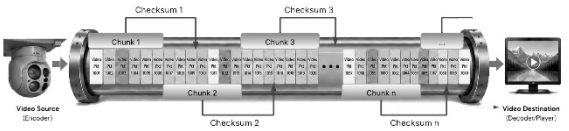
First is the frequency at which the integrity data is inserted. It's tempting, and indeed quite elegant, to calculate and sign the hashes after a predefined number of frames or some external measure, such as after a certain elapsed time (e.g. every second).

However, it is not necessary to adhere to those yardsticks; since the integrity data is only concerned with the stream being altered, there is no drawback to, e.g., calculating the hashes over a defined number of MPEG frames.

The bigger consideration is the processing overhead. Calculating the hash is a relatively low-cost operation and is a pure function of the number of bytes get processed, but signing the hashes is a significantly more expensive task (although see the comment above with regard to PQC-safe ML-KEM algorithm's processor efficiency compared to that of the older standards).

Second is the question of exactly where and how, in the digital video stream, the integrity data should be inserted.

The simplest approach is to just drop in the integrity data as a blob, e.g. after every megabyte of signal data. While perhaps clumsy, it has the advantage that the integrity information is totally distinct from the signal information. The major drawback is that the bitstream is no longer a coherent MPEG Transport Stream, so cannot be displayed or processed without first removing



Integrity Data Embedded in Stream

the added blocks of data.

A better approach would be to include the integrity data within the defined Transport Stream structure. A drawback to this idea is that it will change the overall stream structure slightly (e.g. adding a new elementary stream requires an additional entry to the Program Association Table, PAT). However, the structure of the Transport Stream is itself a framework around the evidentiary data proper, so some level of tolerance to infrastructure pieces will always be required.

An obvious candidate location for the integrity data would be to use the metadata channel (if there is one or creating one if there is not). The KLV encoding scheme supports "value" lengths up to nearly 2^{64} bytes, which is dramatically more than could be required for any hash/signing scheme (2000 bytes or nearly 2^{11} is probably a useful upper bound).

An alternative might be to insert the blob of integrity data into an unused audio channel (with some minimal formatting to conform to the rules for those channels). This may seem like an odd choice, but transferring audio data is a "solved problem", so this concept can sometimes be a "quick and dirty" solution.

The only constraint is really that the location of the identity data (and its structure, come to that) must be known by all relevant parties.

The last major design decision concerns what data to include in the hash. Obviously, one will want to include the video elementary stream, and possibly the audio ones. Frequently, it will be appropriate to also include the KLV metadata elementary channel.

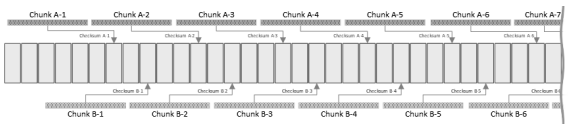
But are the blocks of integrity data included or excluded? As the diagram above shows, the signed checksums are inserted some interval after the relevant data from which the hashes are calculated, so overlapping the checksum for block *n* with the calculation for block *n+1* has clear benefits.

Another consideration here is whether to add additional information to the integrity data block, such as the start and end positions of the data being hashed (perhaps as a byte or time offset into the stream). This ensures that even if the signal data is *identical*, the inclusion of the position ensures that the hashes are not.

Overlapping Signed Hashes

Possibly the most secure approach is to use a pair of overlapping signed hashes, to create alternating integrity blocks.

This approach ensures that nothing can be inserted between the end of one chunk of data (covered by checksum *n*) and the start of the next (covered by checksum *n+1*).



Interlocking Checksums

While it is entirely possible that, with careful design, a single set of checksums can provides the same degree of integrity validation, using the alternating sets ensures that the integrity is not only assured but can be easily seen to be assured.

Conclusion

In no small part because of the abilities of AI tools, the integrity of streaming data is no longer to be taken for granted. This paper has discussed video applications, with a focus on the use of video in legal matters, but the techniques mentioned apply just as well to other packetized data, such as serial streaming telemetry.

Adding cryptographically signed hashes assures both the integrity of the signal, but also confirms the origin.