INTEGRATING FORENSIC WATERMARKING INTO ADAPTIVE STREAMING WORKFLOW

Alexander Giladi

Comcast, USA

ABSTRACT

Per-session forensic watermarking is a process of embedding a unique identifier for each streaming video session into the video content. This identifier will remain in this content, even if the content undergoes multiple cycles of camcording, filtering and transcoding. Forensic watermarking in adaptive HTTP streaming service is a two-step process, and thus the standard workflow is altered in two disjoint points. In the first step, pre-processing, some segments are duplicated and imperceptibly marked, creating two variants of the same segment. These variants are perceptually identical, but contain different bits of information. At the second step, a unique sequence of A and B variant segments is generated per each session, and this list is translated into the manifest format of choice, such as DASH MPD or Apple HLS. This stage is referred to as embedding. This paper will discuss capacity, storage and visual quality trade-offs in selecting a workflow stage at which pre-processing is performed. The two widespread options are (1) a pre-processing step in baseband video, prior to encoding or (2) encoder independent in the compressed-domain. This paper discusses and quantifies ways of reducing capacity and the storage impact of forensic watermarking in both approaches. It will further explore robustness and security implications of the two approaches with respect to different potential vulnerabilities, such as collusion. Lastly, this paper will cover approaches for enabling per-session embedding compatible with existing DVB and DASH-IF compliant MPEG DASH clients.

INTRODUCTION

Digital rights management (DRM) and conditional access systems (CAS) are often seen as the only content security mechanism needed for premium content. However, the main goal of a DRM system is to prevent unauthorized playback of the content. It is not intended to protect from unauthorized distribution of the presented content. Content can be recorded at playback from HDMI output or by camcording. Content can also be obtained via a DRM failure, such as a key compromise or a wholesale compromise of the DRM system.

When unencrypted content is distributed, rights owners have little or no visibility into its distribution. Forensic watermarking allows precise identification of the viewing session from which this content was obtained in its unencrypted form. This is why watermarking is increasingly viewed as an important content security mechanism for premium content, such as early release and UltraHD movies. The MovieLabs Enhanced Content Security specification explicitly requires robust forensic watermarking as a part of its content security suite of technologies.

Watermarking embeds identifiable information (*watermark identifier*) into video signal in a way that (a) is imperceptible to a human viewer, (b) allows precise identification of each viewing session, and (c) is robust enough to survive multiple iterations of transcoding, image processing, and camcording, while maintaining its veracity.

There are several technical approaches to watermarking. A more traditional one-step watermarking embeds a single watermark identifier into a single complete encoded asset. This baseband video modification step (*watermark pre-processing*) can occur either at the server side (i.e., at the content preparation stage) or at the client side in a secure video pipeline. Unfortunately, both one-step approaches are ill-suited for per-session forensic watermarking.

When per-session precision is needed, the server-side one-step approach will result in a single asset encoding per session. This approach results in extreme overheads from both storage (asset per customer) and processing (encode per session) standpoints, and is not practical at scale.

Client-side one-step watermarking scales extremely well, as all the processing happens on decoded baseband video within the secure video pipeline during playback. This requires a tight integration with both hardware and DRM vendors, which is feasible for the set-top box distribution model. Device heterogeneity makes this model much harder to achieve in the adaptive streaming world. Significant device reach implies support for a rapidly changing mix of devices, chipsets and DRMs. Not all of these devices will have hardware-level support for watermarking.

An alternative two-step watermarking approach is based on an observation that in adaptive streaming media segments are the minimal playable entities. Every asset is conceptually a concatenation of media segments, defined in the manifest. Hence each media segment can be treated as a single bit, and for each segment, two encodes (variants) are prepared. They are typically referred to as A and B, but are easier to visualize as ‘0’ and ‘1’ variants. More formally, 2^N variants are needed to express N bits per segment, but N=1 is commonly used. A manifest manipulator will then create a unique manifest per session, and generate a unique sequence of A/B variants per each session on the fly.

The two-step approach is uniquely suited for adaptive streaming deployments. Server-side one-step watermarking will require a separate encode for every single viewing session. This means that its computational and storage complexity is linear to the number of viewing sessions of an asset. The first step of a two-step approach at most doubles the computational and storage overhead, but this overhead is constant and does not depend on the number of viewing sessions. The second step, generation of a unique manifest, is linear to the amount of sessions, but its runtime is negligible, especially compared to other delays inherent in starting a viewing session.

The overhead of the two-step approach is, in our opinion, a small price to pay for a device-independent watermarking technology, and this approach is currently best suited for adaptive streaming deployments from the technological standpoint. This paper will review several issues related to integration of two-step watermarking into an adaptive streaming deployment.

PREPROCESSING CONSIDERATIONS

Pre-encoder domain vs compressed domain

Watermark preprocessing is a transformation of a single raw video input into two raw video outputs. Preprocessor typically modifies only a small percentage of frames.

The most natural place for this preprocessing to occur would be just prior to the encoder, as a part of a preprocessing filter chain. Watermarking algorithms are robust enough to withstand resizing, therefore in multi-rate multi-resolution encodes, watermark preprocessing can run before resizing. This way watermark preprocessing runs once per asset, rather than once per each resolution.

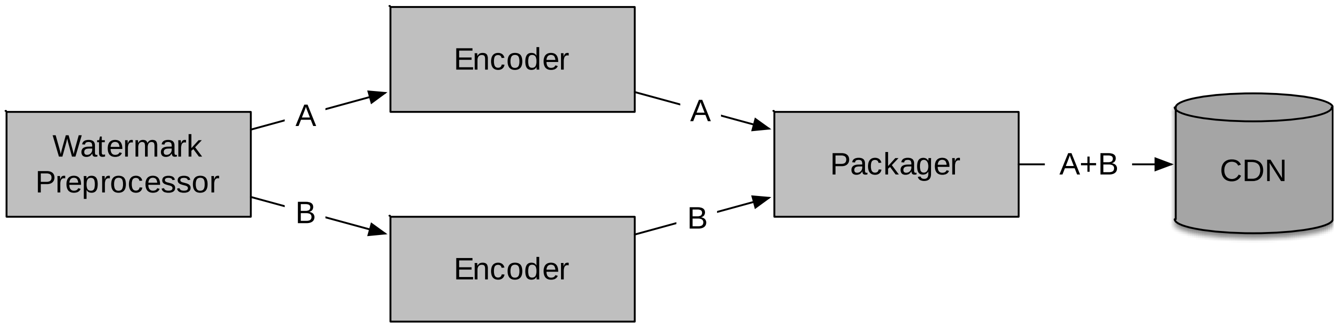

Figure 1. Pre-encoder preprocessing

The obvious downside of the pre-encoder approach is the need for encoder integration, which may be a per-vendor effort. A much easier approach is doing “compressed domain” preprocessing, where preprocessing is applied to a stream, which is already compressed. Typically, this involves decoding the complete segment, preprocessing and encoding the changed part. Since only a small number of frames need to be modified by the preprocessor, many implementations make a shortcut by altering re-encoding only non-reference B frames.

By definition a non-reference frame is not used for prediction by any other frame, so any change within a non-reference frame would not propagate further.

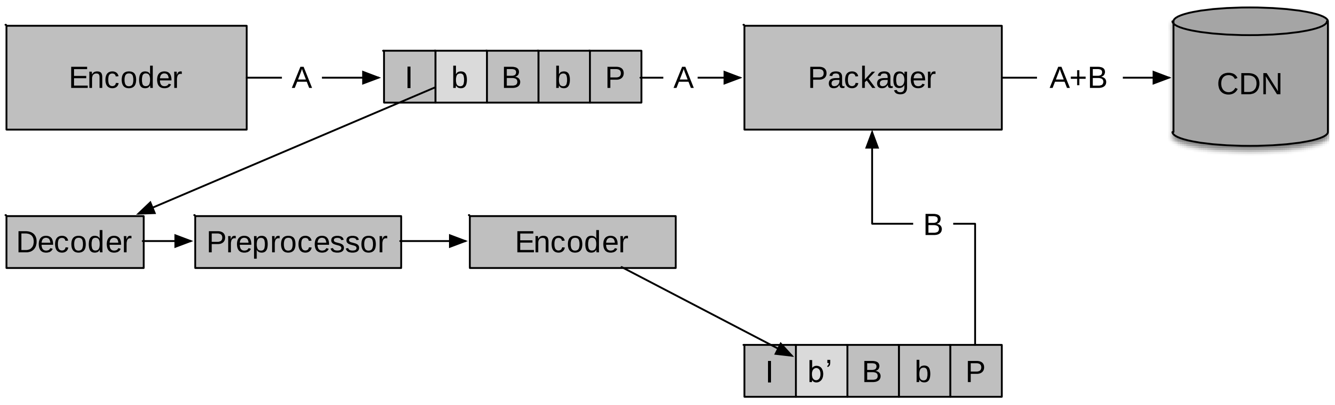

Figure 2. Compression-domain preprocessing with non-reference frame marking

There are several methods to weaken compressed domain watermarking. If the non-reference frame approach is taken, removal of a non-reference frame would completely remove the watermark. In case of premium HFR (high frame rate) UltraHD content, removal of all non-reference B frames will not render content unwatchable: in a typical IbBbP[1] GOP (group of pictures) structure, removal of non-reference B ('b') frames will only halve the frame rate. Non-reference B frame removal can be achieved even in encrypted domain when used with ISO-BMFF (ISO Base Media File Format) and Common Encryption (CENC). CENC only encrypts parts of the VCL (video coding layer) NAL units, so identifying non-reference B frames from the unencrypted parts of the segment is fairly straightforward. This way, watermark removal in case of non-reference B approach can be achieved in encrypted domain in man-in-the-middle scenarios. The use of HLS full-segment AES-128 encryption prevents this approach in encrypted domain. Regardless of container format, this approach is trivial once content is available in unencrypted form (e.g. due to a key compromise).

A different weakness inherent in the non-reference B frame approach is reduced resilience to collusion. Using several encoded copies of a watermarked asset, it is possible to identify differing non-reference frames and remove only them. This removal can be trivially achieved in either encrypted or unencrypted domain. A slightly more complex version of this approach can be used to weaken the watermark from any high-quality capture (e.g. from HDMI). It is possible to measure PSNR (peak signal to noise ratio) between each pair of aligned frames from different sessions. This PSNR is expected to be very high across identical frames, thus an anomalously low PSNR difference between two aligned frames is a strong indicator of a watermarked frame. Since the watermarked frame is not used as a reference in other frames,

[1] The IbBbP notation indicates an IDR ('I') frame followed (in presentation order) by a non-reference B ('b') frame, a reference B frame ('B'), another non-reference B, and, eventually, a P ('P') frame.

the pattern is very easy to identify. Once a frame is identified as a watermarked it can be dropped or replaced.

A more robust alternative to the non-reference approach is using reference frames (e.g. intra-coded frames) and re-encoding the complete segment. While strengthening the overall robustness, the approach results in major or complete re-encoding of the GOP. The results are both an inevitable loss of video quality and very significant increase in computational complexity. Since preprocessing is done post-encoder, re-encoding is needed per each encoder video output (i.e., per each bitrate).

The weaknesses identified above may make compressed domain approach less suited for threat models in which collusion or access to recorded media segments are likely.

Tightening encoder integration

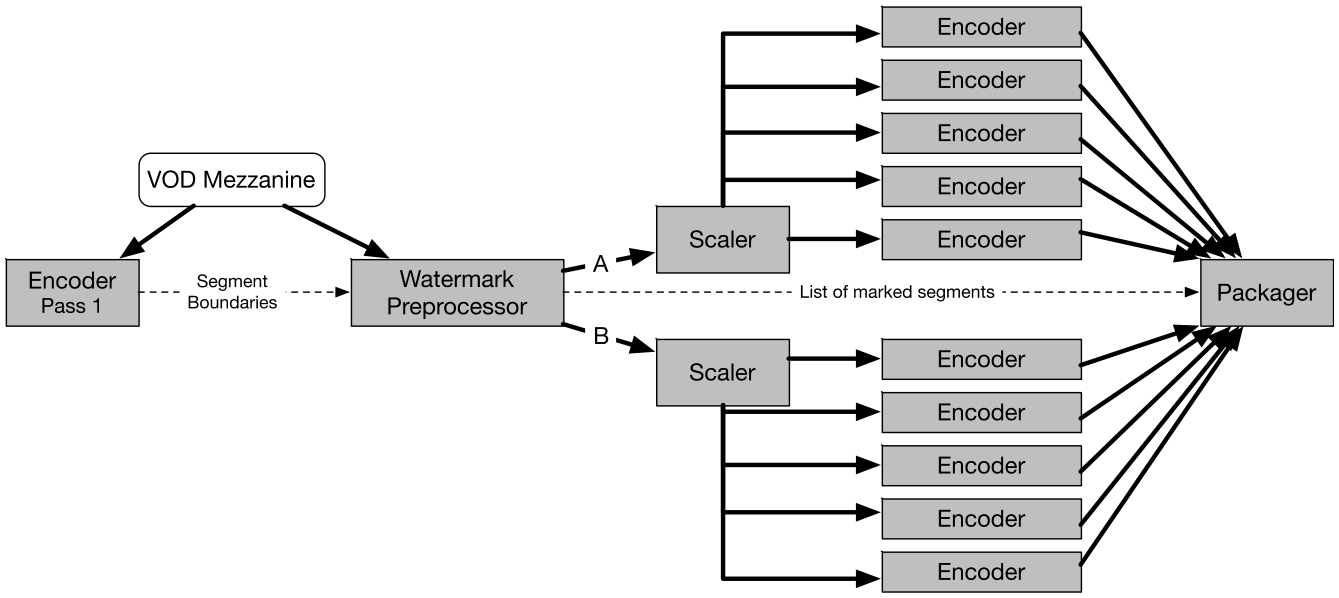

Figure 3. Possible workflow for multi-rate video on demand scenario

When watermark preprocessing is a part of the pre-filter chain, the preprocessor does not have any information regarding segment boundaries. It needs to know segment boundaries in order to know which segment is being modified.

Segments in adaptive streaming deployments always contain an integer number of GOPs. When shorter segments are used (e.g. 2 sec), a GOP is equivalent to a segment. In many cases, constant GOP duration is used in the context of adaptive streaming. This allows a shortcut where segment boundary can be derived from the frame count. For example, in 24fps content, 2-sec fixed GOPs imply that every 48th frame is a segment border.

Variable GOP duration allows better video quality, as it favors putting intra frames at scene change boundaries rather than at fixed intervals. Once variable GOP and segment durations are used explicit information on segment boundaries is beneficial.

In the vast majority of cases, video on demand is encoded using a two-pass encoding process. When two-pass encoding is used, the first pass produces analysis information, which will be used as a basis for final rate control and mode decisions by the encoder at the second pass. The most basic first-pass decision is determining frame type.

Frame type decision determines IDR locations, which, in turn, implicitly determine segment boundaries. This way the preprocessor can relatively safely assume that a segment it needs to watermark is started by an IDR frame. With that said, the most robust approach would be providing explicit segment boundary information to the preprocessor out of band.

Storage overhead

The simplest way of storing two segment variants is by storing them as two separate segments. While this approach has the biggest storage overhead, it also keeps CDN distribution simple: any published segment is available through the CDN, and there is no need for any edge processing.

Some watermarking approaches are resilient enough to create variants for some percentage (e.g. 20%) of the segments. There is a trade-off between the time needed for extraction of the watermark identifier and the percentage of variant segments. The higher the percentage of marked segments, the faster the watermark identifier can be recovered. The lower this percentage is, the lower the overhead is in terms of encoding and storage, and the longer it takes to recover the identifier.

A more efficient approach is storing only differing frames. The fact that only a few frames in a segment are modified makes this approach more efficient, especially when all of them are non-reference B. In this case, storage overhead may even be lower than 5%. This approach was specified in ISO/IEC 23001-12, an international standard defining carriage of variant samples for ISO-BMFF files. MPEG is currently extending this specification to cover carriage of variant samples in MPEG-2 transport streams as well.

Obviously, segments which contain both A and B variants cannot be transmitted directly to the streaming client. Just-in-time edge processing is needed to remove one of the two sample variants for each segment request. While the processing itself is straightforward, there may be an impact on the overall scalability of the system.

EMBEDDING CONSIDERATIONS

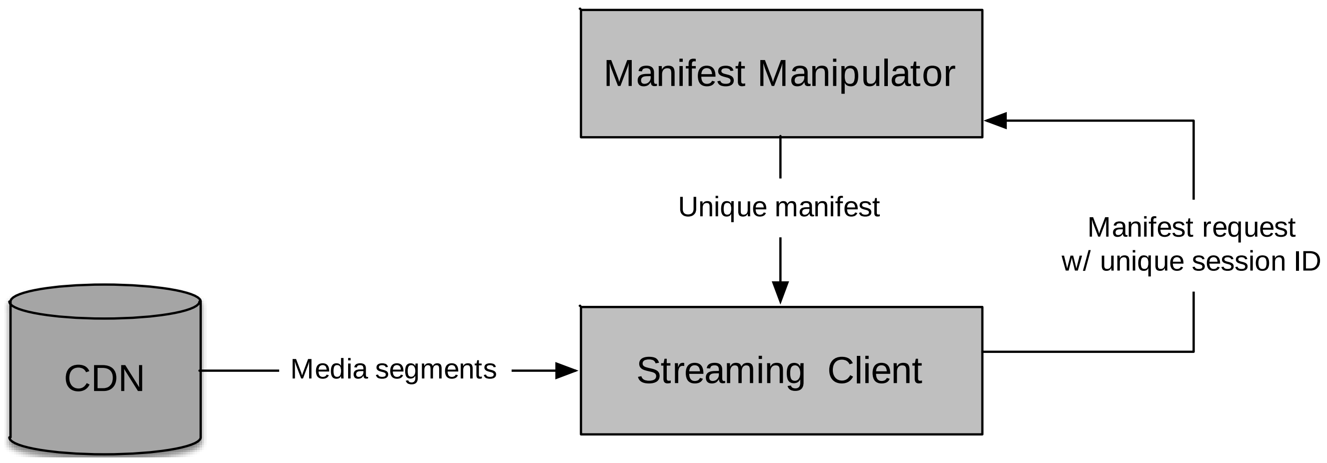

Figure 4. Embedding step

Watermark identifier space

The second step of a two-step forensic watermarking workflow is embedding – generation of a unique session-specific segment sequence in response to an opaque session identifier. If a sequence of segments is viewed as a sequence of bits, longer bit sequences require a longer amount of playback time to recover the sequence completely. Thus, use of reliably unique stateless session identifiers such as UUIDs and cryptographic hashes may negatively affect the overall system performance. On the other hand, short identifiers may result in uniqueness issues or require tight synchronization to keep track of assigned identifiers.

If we look at the identifier space of a viewing session, we can clearly partition it into at least four domains: asset, time, subscriber and device. At the very least, asset identifier is not expected to change across any viewing sessions of a single asset. This suggests a hybrid approach: use of a server-side one-step watermarking approach to embed an asset watermark, followed by a two-step process creating the media segments. This means that asset identifier is encoded identically into every variant, thus a smaller identifier space needs to be expressed via a unique segment sequence. This reduction of identifier space is inexpensive – it has neither bandwidth nor storage overhead, and its impact on encoding time should not be overly significant.

DASH integration

The final result of the embedding step is a session-specific unique sequence of segments. In most products, this is described as a manifest generation; however, conceptually embedding can be thought of as a function returning a string of variant decisions (e.g. “ABBBAAB”). This sequence can often be obtained on a per-segment (“which segment *n* do I provide for session 42?”) or per-session (“what is the unique sequence provided to session 42?”) basis.

The simplest and most straightforward integration happens when discrete segments are used. In this case, each segment is addressable by a URL, and the variant decision string is translated into a unique sequence of URLs. The same sequence will be translated into all variants (HLS) or into all representations in an adaptation set (MPEG DASH) of the same content.

Two-step watermarking can be vulnerable to URL guessing. Guessing the URLs of the A/B variant segments makes is possible to decide which variant to download. This approach is feasible when segment naming convention is predictable – e.g. if there is a “segment_0042A.ts” and “segment_0043B.ts”, it is relatively simple to attempt to download segment_0043A.ts” instead. URL guessing is trivially avoided by using randomized segment URLs, e.g., by embedding UUIDs or salted cryptographic hash into file names (and, consequently, into segment URLs). This way, using GUIDs, the segment names above can become “24dcdceb-fad7-4f90-829b-659ade34255e.ts”, “23759ef0-e216-4799-b57a-7da6c2fb7133.ts”, and “dca62dd8-1201-4088-b51b-fb3987343faa.ts”

This manifest embedding approach maps well into the way HLS works, as m3u8 playlists list each segment individually. It can work the same way with MPEG DASH, as DASH has a SegmentList mode, which is essentially a playlist per representation.

Unfortunately, the SegmentList approach is more helpful for claiming DASH support (“we support Main Profile of DASH”) than for actual integration with an existing DASH ecosystem. The SegmentList addressing mode is not supported in commonly implemented profiles (Live and On Demand) and in specifications by bodies such as DASH-IF, DVB and SCTE. This means that support by commercial DASH ecosystem is not guaranteed.

From the technical standpoint SegmentList is very inefficient. The generated MPD has to list all segments of all representations in a single XML file (as opposed to multiple text files in HLS). Having tens of thousands of XML elements (50K elements is a reasonable number for a two-hour asset with 14 representations and 2-sec segments) results in large manifest sizes and longer XML processing times. Moreover, SegmentList addressing sacrifices one of the major strengths of DASH – templates.

DASH templates make it possible to predict a URL of a segment based on a naming convention, rather than list all segments explicitly. This is extremely efficient when predictable naming conventions are used. For example, instead of explicitly listing “segment_00001.mp4”, “segment_00002.mp4”, “segment_00003.mp4”, and up to “segment_99999.mp4”, DASH templates let one list “segment_$Number%05d$.mp4”. This looks and works like a printf statement.

The problem with DASH templates and watermarking is lack of predictability in segment names. There are two per-segment substitution variables defined in the DASH specification: $Number$ and $Time$. $Number$ is incremented by one for each segment, and thus is unsuitable for our purposes. $Time$ represents precise segment start time, and allows specification of names on a per-segment basis.

Precise time in the $Time$ variable is expressed in ticks of a clock defined in the DASH MPD. This clock is only used for segment download – segment playback is controlled by the time expressed in the segment itself. This allows use of a very high precision clock, with its lower bits used to express the variant. For example, in case of 24fps content and 2-sec fixed-duration segments, 1Hz clock suffices, and frame precision requires a 24Hz clock.

In the example above, it is possible to use a 24KHz clock to express the same timestamps. At the very least, the timestamps would be 1000 clock ticks apart. This 1000-tick space can be used to express variant: for example, variant A can be defined as always starting at its real start time, and variant B can start 1 ms later. Thus, variant A segments can be segment_000000000.mp4, segment_000480000.mp4, while variant B segments will be segment_000000001.mp4 and segment_000480001.mp4. At the point when variant B is inserted, segment duration can be adjusted to reflect the “new” start time. This approach allows compatibility with MPEG DASH Live profiles and, consequently, DASH-IF, DVB and SCTE. The latter makes it possible to use off-the-shelf DASH clients and make minor modifications to manifest manipulator.

The 0ms and 1ms offset is very easy to guess, so it would be prudent to randomize the offsets.

The previous discussion focused on use of multiple segments. Especially in case of MPEG DASH, storing an asset as a single file (single segment) and accessing byte ranges within it (subsegments in DASH terminology) is a very common implementation for video on demand. This approach is codified in the VOD profile of MPEG DASH. In case of DASH, the byte ranges appear in a per-representation index (`sidx`), which is downloaded by the client prior to the playback start.

Byte range addressing is problematic in the context of two-step watermarking. When variant A and variant B are encoded, at least some access units will always have different sizes. This implies different segment sizes of two variants of the same segment. The major issue here is that after the first variant switch, byte ranges no longer be match due to differing subsegment sizes. Let us assume variants A and B of an i^{th} subsegment have sizes $SZ_A(i)$ and $SZ_B(i)$. For i=2 and a string “AA”, the 3rd subsegment starts at byte offset $SZ_A(0) + SZ_A(1)$, while for a string “AB”, the subsegment will start at the offset $SZ_A(i) + SZ_B(i)$. This quickly becomes intractable, as byte range requests will quickly become unique per client, and will almost never correspond to byte ranges in the actual encoded asset.

A simple fix to enable use of byte range addressing is padding variant subsegments to make them have identical sizes. This can be achieved (e.g., by inserting additional SEI messages). In order to avoid giving away the fact that variants exist for the same subsegment or, worse, identifying which subsegment corresponds to A or B variants, all segments should be padded, including those where B variant does not exist. This padding will result in equal sizes, and thus byte ranges will match across all unique combinations of variants. However, the padding approach we described creates a different problem: byte range requests will be identical regardless of the variant request. This renders the manifest manipulation approach useless.

An alternative to manifest manipulation is a variant decision at the CDN edge for each segment request. A client request will reach a transparent proxy at the edge, which will make a per-segment variant decision for a given session. The proxy will then return a subsegment from the corresponding variant. On one hand, this approach significantly impacts system scalability by requiring per-request processing. On the other hand, this approach makes it possible to work with byte ranges; moreover, it hides the very existence of watermarking from the streaming client. An additional consequence can also be reduction in storage – only differing samples or segments need to be stored, as the added cost of just-in-time repackaging is relatively small.

SUMMARY

In this paper, we provided a brief overview of considerations for integrating forensic watermarking into operators' high-value content offerings. We showed why the two-step approach is the optimal approach to forensic watermarking in context of client device heterogeneity.

We showed the implications of frame type selection, especially when collusion is used to weaken the watermarking, and concluded that approaches relying on non-reference frames are less desirable from both security and visual quality aspects.

We further discussed trade-offs between storage and just-in-time format conversion when serving watermarked segments.

We further discussed the size of watermark identifier space.

Lastly, we addressed the issues of integrating two-step watermarking with DASH players supporting Live and VOD profiles of MPEG DASH. We showed that this integration can be done securely and efficiently.

We hope this paper will help solution architects within the industry who are considering implementing forensic watermarking.

2017