DASH-IF IOP Watermarking

Contents

Introduction

This document describes proposed architecture and API for supporting forensic watermarking for Over-The-Top (OTT) on content that is delivered in an Adaptive Bitrate (ABR) format. To the possible extend, the proposed solutions do not make assumptions on the ABR technology that is being used, it can be for example, DASH or HLS.

While digital watermarking can be used for different use cases, this document will focus on forensic use cases. In this context, it is used to define the origin of content leakage. the watermarking technology modifies media content in a robust and invisible way in order to encode a unique identifier, e.g., a unique session ID. The embedded watermark provides means to identify where the media content, that has been redistributed without authorization, is coming from. In other words, the watermark is used to forensically trace the origin of content leakage.

1. Scope

2. References

2.1. Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ISO/IEC 23009-1:2021 Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.

[2] ISO/IEC 13818-1:2019 Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems URL: https://www.iso.org/standard/75928.html

[3] R. Pantos. HTTP Live Streaming 2nd Edition. Internet Draft. URL: https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis-09

[4] M. Jones; J. Bradley; N. Sakimura. JSON Web Token (JWT). May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7519

[5] M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7515

[6] UHD Forum, Watermarking API for Encoder Integration, version 1.0.1, March 2021. URL: https://ultrahdforum.org/guidelines/

[7] The Open Group Base Specifications Issue 7, IEEE, Std 1003.1 2018 Edition, 31 January 2018. URL: https://pubs.opengroup.org/onlinepubs/9699919799/

2.2. Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Edge Services: Watermarking, Overview for Watermarking Partners. Version 1.9, July 2020. URL: https://github.com/akamai/media-delivery/blob/master/Integration%20Specs/Akamai%20AMD%20Edge%20Watermarking%20Specification.pdf

3. Definition of Terms, Symbols and Abbreviations

3.1. Terms

For the purposes of the present document, the following terms apply:

Client-driven watermarking: The action of watermarking content when the user device is performing some actions allowing it to make unique requests for content. The user device embeds a watermarking agent that is integrated with the application.

Client-side watermarking: The action of watermarking when the user device is the sole responsible for doing the actual watermarking of content. The user device embeds a watermarking agent that is integrated with the audio-visual rendering engine.

Server-driven watermarking: The action of watermarking content when the user device is not performing any other operation than conveying information such as tokens, between servers that are responsible for doing the actual watermarking of content that is delivered to the user device.

Variant: A Variant is an alternative representation of a given segment of a multimedia asset. Typically, a Variant is a pre-watermarked version of the segment.

Watermark (WM) pattern: A series of A/B decisions for every segment that is unique per user device.

3.2. Symbols

Void.

3.3. Abbreviations

For the purposes of the present document, the following abbreviations apply:

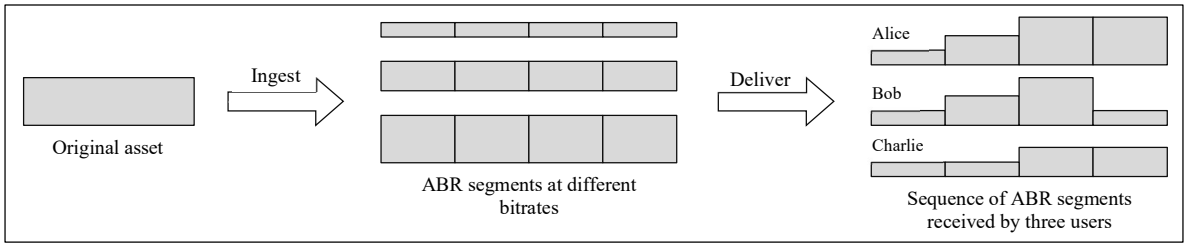
ABR	Adaptive Bit Rate
AES	Advanced Encryption Standard
AF	Adaptation Field
AVC	Advanced Video Codec
CDN	Content Delivery Network
CMAF	Common Media Application Format
DASH	Dynamic Adaptive Streaming over HTTP
DRM	Digital Rights Management
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISOBMFF	ISO Base Media File Format
JSON	JavaScript Object Notation
JWT	JSON Web Token
MPD	Media Presentation Description
OTT	Over The Top
RIST	Reliable Internet Stream Transport
RTMP	Real-Time Messaging Protocol
RTP	Real Time Protocol
SEI	Supplemental Enhancement Information
SRT	Secure Reliable Transport
TS	Transport Stream
TV	Television
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VOD	Video On Demand
WM	Watermark
WMID	Watermark Identifier
WMT	Watermark Token

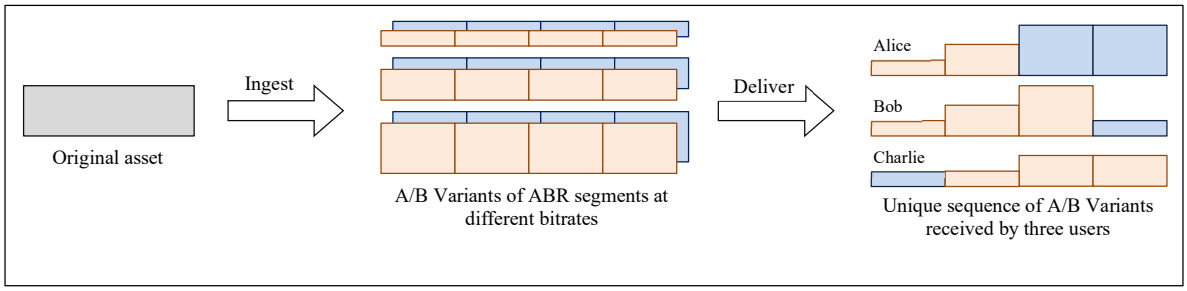
4. OTT Watermarking Using Variants

The objective of forensic watermarking is by definition to deliver a unique version of a media asset to the different users consuming the asset. This is somewhat in opposition with media delivery mechanisms that aim at delivering the same asset to all users for efficiency purposes. As a result, in the broadcast era, a typical approach was to perform the watermarking operation at the very last step of the media delivery pipeline, that is to say within the end user device e.g., a set-top box. This solution has the virtue of leaving the whole media delivery pipeline unaltered but raises security and interoperability challenges when a large variety of devices owned and operated by the end user shall be supported. This is for instance the case with over-the-top (OTT) media delivery where content is consumed on mobile phones, tablets, laptops, connected TVs, etc. As a result, new forensic watermarking solutions have gained momentum that do not perform security-sensitive and complex operations in the end user realm. While such approaches require minimal changes in the end-user devices, they do mandate the media delivery pipeline to be modified accordingly.

A notable example of such network-side watermarking solutions is OTT watermarking using Variants for adaptive bitrate (ABR) content. In this case, the content is delivered by segments. The baseline idea is then to generate pre-watermarked Variants of each segment and to modify the delivery protocol so that each end user receives a unique sequence of Variants depending on a watermarking identifier (WMID) that has been assigned to the end user. The semantic of this WMID is context dependent and can be, for instance, a device identifier, an account identifier, a session identifier, etc. Figure 1 illustrates a particular case of this strategy, coined as A/B watermarking, where there are two Variants generated for each segment, each Variant containing a watermark that either encodes the information ‘0’ or ‘1’. As a result, the watermarking system will require the transmission of a sequence of Variants as long as the length of the WMID to successfully recover the whole WMID.



(a) ABR content delivery



(b) A/B Variants for ABR content delivery

Figure 1: A/B watermarking concept.

When using pre-watermarked Variants, the serialization process essentially boils down to delivering a unique sequence of Variants to each individual end user. There are two main families of methods to achieve this:

1. Server-driven methods, wherein the client does perform no operation related to watermarking. It simply fetches and forwards digital objects, e.g., some tokens, to the CDN that is responsible for delivering a unique sequence of Variants.

2. Client-driven methods, wherein the client is responsible for the serialization operation. For instance, it relies on some session-based digital object to tamper the URI ABR segments and thereby directly query a unique sequence of Variants from the CDN.

This document is describing the server-driven methods. Client-driven methods are not part of this specification.

5. Server-Driven Architecture and Workflows

5.1. Introduction

In the server-driven architecture, the device is unaware that content it consumes is watermarked. The device only exchanges a token with servers allowing these servers, usually CDN edge servers, to make the decision on which A or B Variant it delivers to the device. This document defines metadata that removes the need for naming conventions and assumptions by allowing the watermarking process to send metadata all the way to the edge to enable the sequencing of bits. The following goes through the functional architecture and describes the workflows when preparing content and when consuming content.

In the following, it is assumed that the edge server is a CDN edge. There are optional architectures, but this does impact the overall functional architecture and workflows. In addition, all the workflows are only examples of possible implementations.

5.2. Functional Architecture

Figure 2 shows the simplified high-level functional architecture and the different interaction between the components that are involved in the flows when a device consumes watermarked content. Note that this also assumes that content is encrypted, as this is very likely that watermarking is added for premium content that is also encrypted and protected by a DRM system.

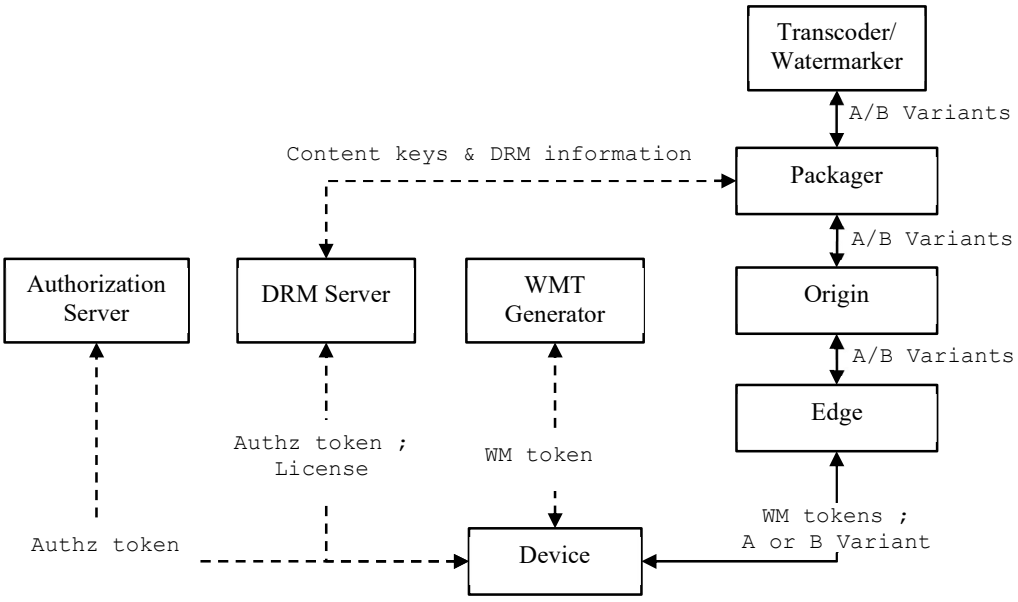


Figure 2: Functional architecture.

To consume content, a device needs, at minimum, to have an authorization token (for getting a DRM license) and a WM token that contains a WM pattern, a series of A or B decisions. The device is responsible for obtaining the required data before requesting segments to the CDN.

5.3. Content Preparation

5.3.1. General Principles

Content preparation means the generation of A/B Variants and the push of content on the origin. It is under a workflow manager responsibility in case of VOD and fully automated for Live content. The transcoder generates the different watermarked (A and B) Variants of the adaptive content. The encrypted segments, the DASH manifest and HLS playlists are generated by the packager. Manifests and playlists, delivered to the devices, are not personalized, meaning these are the same for all devices and only contains URLs for the segments without any tokenization per device for

example. It is at consumption time, that the right URL is resolved for each device and each segment by the edge server. A simplified flow is shown in Figure 3 for the case of Live content assuming that the DASH-IF ingest protocol is used (note that content protection steps are omitted for clarity). For encrypted content, Variants of every segment part of the same Representation may be encrypted using the same encryption method and with the same content key, meaning the same DRM license allows decrypting the A and B Variants. In addition to the Variants, the transcoder pushes also structures called `WMPaceInfo` that contain information allowing the packager and the origin servers to properly associate the pieces of Variants that are pushed to a bit position on the WM pattern (see clause 5.3.2). In such flow, the packager can aggregate multiple ingest segments into one egress segment, with the limitation that only ingest segments carrying the same `pos` value (see clause 5.3.2) can be aggregated together. The Packager shall not aggregate segments that have inconsistent `WMPaceInfo`.

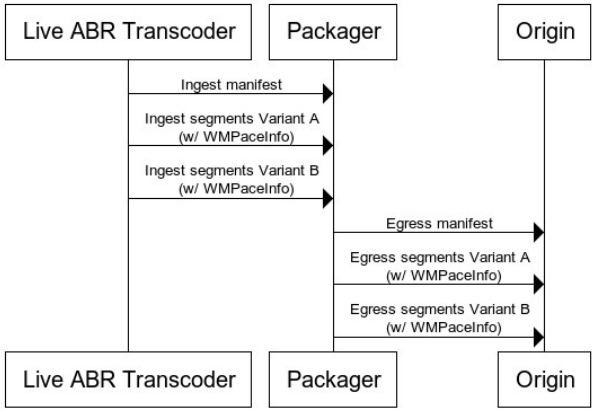


Figure 3: Example of Live DASH content preparation workflow using the DASH-IF ingest protocol.

In order to ensure interoperability, one of the following identification scheme shall be used for the identification of the Variants:

- A lower case letter beginning with a. Variants are then a, b and so on.
- A number beginning with 0. Variants are then 0, 1 and so on.

If both schemes are used in a deployment, variant a is equal to variant 0, variant b is equal to variant 1 and so on.

5.3.2. WMPaceInfo Data

When a device requests a segment, the edge server needs to know which bit in the unique WM pattern it has to consider in order to retrieve either A or B Variant of the requested segment before delivering it to the device. The `WMPaceInfo` structure contains this mapping. It is transmitted between the transcoder to the following servers that may need it (origin, packager servers for example).

`WMPaceInfo` structure is as shown in Table 1.

Table 1: `WMPaceInfo` structure.

Attribute	Producer	Consumers	Purpose
`iswm`	Transcoder	Packager, CDN	
`variant`	Transcoder	CDN	Integration, debugging
`pos`	Transcoder	CDN	Bit position in the WM pattern
`firstpart`	Transcoder	Packager, Origin	Egress packaging
`nbpart`	Transcoder	Packager, Origin	Egress packaging

The attributes are:

- `iswm` states whether or not watermark is applied on content referenced by this `WMPaceInfo` data. `iswm`=false indicates that content is not watermarked. `iswm`=true indicates that content is watermarked. When `iswm`=false, `variant` do not carry relevant information and `pos` shall be equal to 0.
- `variant` gives the Variant identification, 0, 1 and so on. This information can be useful up to the edge server for verifying that the right Variant has been obtained.
- `pos` is the index in the WM pattern to consider for this segment. Positions are zero-based. For example, `pos`=33 indicates that this particular segment refers to position 34 of the WM pattern.
- `firstpart` informs whether or not this segment is the first one with this `pos` value. It is equal to true if this is the case, otherwise it is equal to false.

- `nbpart` is the number of consecutive segments with this `pos` value. `nbpart` shall be consider as an upper boundery of the number of segments to be aggregated. For example, in case of ads insertion, a segment may have `firstpart`=true even if less than `nbpart` segments have been received because of the early termination of the segment before the ads. In this case, the value of `firstpart` takes precedence over the value of `nbpart`.

5.3.3. Conveying WMPaceInfo

5.3.3.1. Introduction

`WMPaceInfo` is delivered from the transcoder to others servers with the same protocol than the segments. There is no unique mechanism for this. This document does not recommend one preferred option applicable for all protocols, Table 2 only present some possible options for conveying `WMPaceInfo` with a preferred option for some protocols (in bold in the table). The following goes through these different options.

Table 2: Possible options for conveying `WMPaceInfo` information.

Ingest protocol	WMPaceInfo delivery options
RTMP	SEI
RTP/UDP/RIST/SRT	SEI, TS adaptation field
HLS/TS over HTTP POST	**HTTP header**, SEI
CMAF-based protocols/formats (HLS/fMP4, DASH) over HTTP POST	**HTTP header**, ISOBMFF box, SEI
File access protocol	ISOBMFF box, SEI, **sidecar file**

5.3.3.2. HTTP Header

This data is provided in the response header when a segment is requested by a server or as part of the request header when content is pushed over HTTP.

As shown in Figure 4, as an example, the `WMPaceInfo` header field is created by the transcoder, when exactly one segment is ingested to the following server (packager, or origin). Each ingested segment has its own header. Every server keeps the information within the header associated to the ingested segment. In some cases, for example when the origin does additional packaging, the header may be updated.

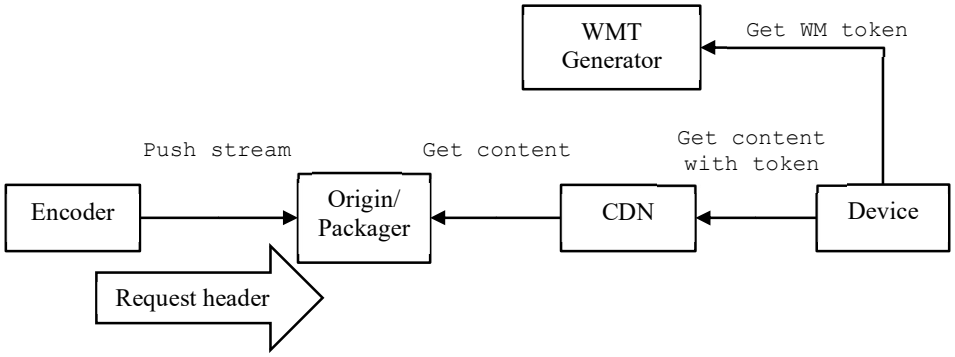


Figure 4: `WMPaceInfo` header information carriage in HTTP headers.

The `WMPaceInfo` header field contains a JSON object with the following format:

```
{
  "version":   version,
  "iswm":      iswm,
  "variant":   variant,
  "pos":       pos,
  "firstpart": firstpart,
  "nbpart":    nbpart
}
```

Where

- `version` is set to 1 for `WMPaceInfo` compliant to this document.
- `ismw, variant`, `pos`, `firstpart` and `nbpart` are defined in clause 5.3.2.

Below is an example of the JSON element added in the header where the payload of the HTTP request or response contains the full segment of Variant A.

```
{
  "version": 1,
  "iswm": true,
  "variant": 0,
  "pos": 33,
  "firstpart": true,
  "nbpart": 1
}
```

5.3.3.3. ISOBMFF Box

The format of `WMPaceInfo` shall be as follows

```
class WMPaceInfo {
  unsigned int(8)  version;
  unsigned int(8)  variant;
  unsigned int(1)  emulation_1;
  unsigned int(15) pos;
  unsigned int(1)  firstpart;
  unsigned int(1)  emulation_2;
  unsigned int(1)  iswm;
  unsigned int(5)  reserved;
  unsigned int(8)  nbpart;
}
```

Where

- `version` is set to 1 for `WMPaceInfo` compliant to this document.
- `variant`, `pos`, `firstpart`, `iswm` and `nbpart` are defined in clause 5.3.2.
- `emulation_1` and `emulation_2` are set to 1.

Within an ISOMBFF file, the WMPaceInfo data shall be carried in the following box:

Box Type: `'wmpi'`
Container: Top level box
Mandatory: No
Quantity: Zero or one

```
aligned(8) class WMPaceInfoBox extends Box('wmpi')
{
  WMPaceInfo();
}
```

This box should be inserted only at the beginning of a segment in order to facilitate content manipulation when zero-padding it.

5.3.3.4. SEI Message

SEI messages are inserted in the stream with a specific syntax depending on the codec. [6] provides the syntax for AVC, HEVC and AV1 video codecs in Annex B. In these messages:

- The UUID shall be equal to `0xbec4f824-170d-47cf-a826-ce008083e355`
- The watermarking metadata is the `WMPaceInfo` data with the format defined for the class `WMPaceInfo()` in clause 5.3.3.3.

This message should be inserted for the first frame of a segment in order to facilitate content manipulation when zero-padding it.

5.3.3.5. TS Adaptation Field

Following clause U of [2], the format of the private adaptation field descriptor carrying the `WMPaceInfo` data is as follows

Syntax	No. of bits	Mnemonic
`temi_WMPaceInfo_descriptor {`		
`  af_descr_tag`	`8`	`uimsbf`
`  af_descr_length`	`8`	`uimsbf`
`  WMPaceInfo()`	`48`	`uimsbf`
`}`		

Where

`af_descr_tag` – is an 8-bit field that uniquely identifies this AF descriptor. It is equal to [tbd]

`af_descr_length` – is an 8-bit field specifying the number of bytes of the AF descriptor immediately following **`af_descr_length`** field.

`WMPaceInfo()` – is an 48-bit field that carries the information defined for the class `WMPaceInfo()` in clause 5.3.3.3.

This message should be inserted for the first frame of a segment in order to facilitate content manipulation when zero.padding it.

5.3.3.6. Sidecar File

When segments (discrete files or byteranges) are delivered with a file transfer protocol, it may be convenient to have `WMPaceInfo` data in a sidecar file. This file carries the data for all segments of a track.

When segments are discrete files, the sidecar file is an array of JSON objects with the following format.

```
{
  "segmentType":           segmentType,
  "variantSubPath": [
    {
      "variant":           variant,
      "subPath":           subPath
    }
  ],
  "segments": [
    {
      "segmentRegex":      segmentRegex,
      "WMPaceInfoObject": WMPaceInfoObject
    }
  ]
}
```

Where

- `segmentType` defines the method for addressing segments within a track. In case of discrete files, it is equal to "`discrete`".
- `variantSubPath` is an array that contains, for every Variant, its identification under `variant` (0 for Variant A, 1 for Variant B and so on) and in `subPath`, the sub path to be added in the full path for the entry point of the Variant on the origin. See clauses 5.3.4 and 5.5.4.1 for more details.
- `segmentRegex` is a POSIX extended regular expression as described in clause 9 of [7]. It allowing to define the filename of the segments for which the `WMPaceinfoObject` applies.
- `WMPaceInfoObject` is the `WMPaceInfo` data as a JSON object defined in clause 5.3.3.2.

NOTE: Using regular expressions and file naming conventions allows reducing the number of required side car files. The same side car file could be used for all renditions of a given segment. This allows the origin to reduce the number of sidecar files, but the edge will always receive several copies of the same data as caching is done on the xact filename. It is recommended to balance the advantages and disadvantages of regular expressions, because of its CPU load on the origin.

The following is an example for a set of segments where the filenames satisfy the `segmentRegex` expression. Note that in this example, the value of `variantSubPath` contains information for 2 Variants and `variant` (part of `WMPaceInfoObject`) indicates that this example is for Variant A. The data in `WMPaceInfoObject` are for any Variant A where the filename is in the form of `video_segment_[repID]_123.mp4` or `video_segment_[repID]_124.mp4` allowing to have one element for all Representations (for DASH) of the same segment for example.

```
{
  "segmentType": "discrete",
  "variantSubPaths": [
    {
      "variant": 0,
      "subPath": "a"
    },
    {
      "variant": 1,
      "subPath": "b"
    }
  ],
  "segments": [
    {
      "segmentRegex": "video_segment_ .*?_123.mp4",
```

```
      "WMPaceInfoObject":
        {
          "version": 1,
          "iswm": true,
          "variant": 0,
          "pos": 21,
          "firstpart": true,
          "nbpart": 1
        }
    },
    {
      "segmentRegex": "video_segment_ .*?_124.mp4",
      "WMPaceInfoObject":
        {
          "version": 1,
          "iswm": true,
          "variant": 0,
          "pos": 22,
          "firstpart": true,
          "nbpart": 1
        }
    }
  ]
}
```

When segments are byteranges that can be accessed from the DASH manifest or HLS playlist, the file contains an array of JSON objects with the following format.

```
{
  "segmentType":          segmentType,
  "variantSubPaths": [
    {
      "variant":          variant,
      "subPath":          subPath
    }
  ],
  "segments": [
    {
      "startRange":       startRange,
      "endRange":         endRange,
      "WMPaceInfoObject": WMPaceInfoObject
    }
  ]
}
```

Where

- `segmentType` defines the method for addressing segments within this track. In case of byteranges, it is equal to "`byterange`".
- `variantSubPath` is an array that contains, for every Variant, its identification under `variant` (0 for Variant A, 1 for Variant B and so on) and in `subPath`, the sub path to be added in the full path for the entry point of the Variant on the origin. See clauses 5.3.4 and 5.5.4.1 for more details.
- `startRange` and `endRange` are respectively providing the position of the first byte and the position of the last byte that are in the byte range. These are expressed as a byte offset from the beginning of the track.
- `WMPaceInfoObject` is the `WMPaceInfo` data as a JSON object defined in clause 5.3.3.2 that applies for the range defined by `startRange` and `endRange`.

The first byterange of a track that contains the initialisation segment. When segments are delivered with byteranges, it is not possible to differentiate the request for this part of the file from a request for a media segment when using a pattern as described in clause 5.3.4. The sidecar file shall include an entry for every byte ranges. If it is not a media segment, then `WMPaceInfo` associated to this range shall be equal to:

```
WMPaceInfoObject:
{
  "version": 1,
  "iswm": false,
  "variant": 0,
  "pos": 0,
  "firstpart": true,
  "nbpart": 1
}
```

The following is an example of a file with an initialisation segment part of the byterange from 0 to 1117 and two segments.


```
{
  "segmentType": "byterange",
  "variantSubPaths": [
    {
      "variant": 0,
      "subPath": "a"
    },
    {
      "variant": 1,
      "subPath": "b"
    }
  ],
  "segments": [
    {
      "startRange": 0,
      "endRange": 1117,
      "WMPaceInfoObject":
        {
          "version": 1,
          "iswm": false,
          "variant": 0,
          "pos": 0,
          "firstpart": true,
          "nbpart": 1
        }
    },
    {
      "startRange": 1118,
      "endRange": 1701211,
      "WMPaceInfoObject":
        {
          "version": 1,
          "iswm": true,
          "variant": 0,
          "pos": 33,
          "firstpart": true,
          "nbpart": 1
        }
    },
    {
      "startRange": 1701212,
      "endRange": 3490692,
      "WMPaceInfoObject":
        {
          "version": 1,
          "iswm": true,
          "variant": 0,
          "pos": 34,
          "firstpart": true,
          "nbpart": 1
        }
    }
  ]
}
```


5.3.4. Segments Path Structure on the Origin

5.3.4.1. Introduction

The DASH manifest [1] and HLS playlist [3] served to the devices are “neutral”, meaning that

- The same playlist or manifest is served to all devices of all end-users.
- It does not expose different names for A and B Variants of a given segment.

Nevertheless, the segments actually served to the devices need to be either A or B Variant, depending on the WM token information. The media segments path at the CDN edge and at the origin is therefore different.

5.3.4.2. Locating the Variants

Taking the example of a DASH manifest located at https://edge.hostname/path/to/endpoint/index.mpd that references video segments as

```
<SegmentTemplate timescale="60000" media="video_segment_$RepresentationID$_$Number$.mp4"
initialization="video_init_$RepresentationID$.mp4" startNumber="10967120"
presentationTimeOffset="903486496960">,
```

A request received at the CDN edge for https://edge.hostname/path/to/endpoint/video_segment_5_8353305.mp4 shall be translated into a forward request for https://origin.hostname/path/to/endpoint/**a**/video_segment_5_8353305.mp4 or https://origin.hostname/path/to/endpoint/**b**/video_segment_5_8353305, depending on the value extracted from the WM pattern for this segment if there are two Variants and assuming that Variants `subPath` (part of `variantSubPaths`) is a lower case letter. The same logic applies if the watermarking is done through audio segments.

Watermarked objects names shall also include a pattern that the CDN can match to differentiate these objects from non-watermarked objects (initialization segments, subtitles, trickplay images). In the example above, the pattern would be '**video_segment_**'. The pattern is implementation dependant.

Watermarked objects shall include in the sub-path in the CDN forward requests to the origin the value of `subPath/` (part of `variantSubPaths`) defined in clause 5.3.3.6. The exact value of the forward request is defined in clause 5.5.4.1.

As written above, egress DASH manifests and HLS playlists are neutral, but ingest DASH manifests and HLS playlists include information about the A and B Variants being ingested, this is

- The ingest path
- Some signalling elements to describe if a DASH Adaptation Set includes the A or B Variants, or if an HLS media playlist includes A or B Variants.

The ingest of A and B Variants shall be done on the specific ingest paths as defined by values `subPath` (part of `variantSubPaths`), the ingest paths include a `subPath/` for every Variant.

DASH manifests shall include an **`EssentialProperty`** element per Variant. The **`EssentialProperty`** elements are added at the level where the modifications are to be applied (**`AdaptationSet`** or **`Representation`** level). It has the `@schemeIdUri` attribute equal to `http://dashif.org/guidelines/watermarking_variant#<a letter>` where `<a letter>` identifies the Variant this **`EssentialProperty`** element is associated to and `@value` attribute is equal to the path to the Variant identified by `<a letter>`. If there are additional Variant (A, B and C for example), the `@schemeIdUri` attribute is different for each element, for example, for example for Variant C, `@schemeIdUri` attribute shall be equal to `http://dashif.org/guidelines/watermarking_variant#c`. The `@media` attribute from the **`SegmentTemplate`** element or the **`BaseURL`** element shall be replaced by the `@value` attribute value from this **`EssentialProperty`** element for getting the path for the segments of the Variant identified by `<a letter>`. The `@value` element contains the sub-path and shall use the same convention that the `subPath` (part of `variantSubPaths`), meaning that if is a lower case letter, then `@value` shall use a lower case letter.

The following is an example of a DASH ingest manifest with two Variants, A and B, the watermarking signalling is highlighted in red. **`EssentialProperty`** elements gives the path for the segments for Variant A and Variant B. The path given by the `@media` attribute from the **`SegmentTemplate`** element is replaced by the origin for getting the real path to segments. In this example, lower case letters are used. An example with a **`BaseURL`** element is given in clause 5.3.4.3.

```
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1" bitstreamSwitching="true">
  <EssentialProperty schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a"
value="a/video_segment_$RepresentationID$_$Number$.mp4"/>
  <EssentialProperty schemeIdUri="http://dashif.org/guidelines/watermarking_variant#b"
value="b/video_segment_$RepresentationID$_$Number$.mp4"/>
  <SegmentTemplate timescale="60000"
media="video_segment_$RepresentationID$_$Number$.mp4"
initialization="video_init_$RepresentationID$.mp4" startNumber="10967120"
presentationTimeOffset="903486496960">
    <SegmentTimeline>
      <S t="903487696960" d="240000"/>
      <S t="903487936960" d="186000"/>
    </SegmentTimeline>
  </SegmentTemplate>
  <Representation id="27" width="1920" height="1080" frameRate="30/1" bandwidth="5000000"
codecs="avc1.4D4028"/>
  <Representation id="24" width="1280" height="720" frameRate="30/1" bandwidth="3000000"
codecs="avc1.4D401F"/>
```

```
  <Representation id="26" width="640" height="360" frameRate="30/1" bandwidth="1499968"
codecs="avc1.4D401E"/>
</AdaptationSet>
```

For HLS ingest playlists, the master playlist shall include all the A and B Variants with a custom attribute specifying the Variant ('a' or 'b'). The attribute is `WATERMARKING-VARIANT`. A combination of both audio and video watermarking can therefore be used in a single streamset. In the media playlists, the only specific signalling is the segments paths that reflects on which ingest path the Variants are ingested. The sub-paths in the media playlists shall use the same convention that the `subPath` (part of `variantSubPaths`), meaning that if is a lower case letter, then `@value` shall use a lower case letter.

The following is an example of HLS ingest playlists, the watermarking signalling is highlighted in red (this theoretical example, both the video and audio are watermarked). In this example, lower case letters are used.

Master playlist

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-STREAM-INF:BANDWIDTH=5227200,AVERAGE-
BANDWIDTH=3511200,CODECS="avc1.4d401f,mp4a.40.2",RESOLUTION=1280x720,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2719200,AVERAGE-
BANDWIDTH=1861200,CODECS="avc1.77.30,mp4a.40.2",RESOLUTION=640x360,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=8571200,AVERAGE-
BANDWIDTH=5711200,CODECS="avc1.4d4028,mp4a.40.2",RESOLUTION=1920x1080,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="a"
video_3.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=5227200,AVERAGE-
BANDWIDTH=3511200,CODECS="avc1.4d401f,mp4a.40.2",RESOLUTION=1280x720,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_4.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2719200,AVERAGE-
BANDWIDTH=1861200,CODECS="avc1.77.30,mp4a.40.2",RESOLUTION=640x360,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_5.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=8571200,AVERAGE-
BANDWIDTH=5711200,CODECS="avc1.4d4028,mp4a.40.2",RESOLUTION=1920x1080,FRAME-
RATE=30.000,AUDIO="program_audio",WATERMARKING-VARIANT="b"
video_6.m3u8
#EXT-X-IMAGE-STREAM-INF:BANDWIDTH=55649,AVERAGE-
BANDWIDTH=23579,RESOLUTION=308x174,CODECS="jpeg",URI="trickplay_7.m3u8"
#EXT-X-MEDIA:TYPE=AUDIO,LANGUAGE="eng",NAME="Stadium
ambiance",AUTOSELECT=YES,DEFAULT=YES,GROUP-
ID="program_audio",URI="audio_8.m3u8",WATERMARKING-VARIANT="a"
#EXT-X-MEDIA:TYPE=AUDIO,LANGUAGE="eng",NAME="Stadium
ambiance",AUTOSELECT=YES,DEFAULT=YES,GROUP-
ID="program_audio",URI="audio_9.m3u8",WATERMARKING-VARIANT="b"
```

NOTE: While it is a legal signaling in HLS to have multiple `EXT-X-MEDIA` tags with the same `GROUP_ID` value, each tag shall have a different `NAME` value. As these playlists are not for devices to consume and in order to minimize the processing on the playlists, the ingest playlists do not follow this rule and multiple `EXT-X-MEDIA` share the same `NAME` value.

Media playlist (A Variant)

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:11352692
#EXT-X-MAP:URI="video_init_1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2021-09-15T00:48:38.933Z
```

```
#EXTINF:6.000,
a/video_segment_1_11352692.mp4
#EXTINF:6.000,
a/video_segment_1_11352693.mp4
#EXTINF:6.000,
a/video_segment_1_11352694.mp4
#EXTINF:6.000,
a/video_segment_1_11352695.mp4
#EXTINF:6.000,
a/video_segment_1_11352696.mp4
```

Media playlist (B Variant)

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:11352692
#EXT-X-MAP:URI="video_init_1.mp4"
#EXT-X-PROGRAM-DATE-TIME:2021-09-15T00:48:38.933Z
#EXTINF:6.000,
b/video_segment_1_11352692.mp4
#EXTINF:6.000,
b/video_segment_1_11352693.mp4
#EXTINF:6.000,
b/video_segment_1_11352694.mp4
#EXTINF:6.000,
b/video_segment_1_11352695.mp4
#EXTINF:6.000,
b/video_segment_1_11352696.mp4
```

When the ingested content is not watermarked anymore, then

- For DASH content, the **`EssentialProperty`** elements shall be removed from the ingest manifest and a new **`Period`** shall be created.
- For HLS content, the encoder shall create a new master playlist that does not include `WATERMARKING-VARIANT` attributes. It also stops delivering the additional media playlists for the B Variant and others if present.

NOTE: Stopping watermarking content is different from toggling off enforcement (see clause 5.5.5).

5.3.4.3. Locating the Sidecar File

The sidecar file is part of the ingest with the DASH manifest or HLS playlist, the link to this file is added in different places depending on the format.

DASH ingest manifests shall include an **`EssentialProperty`** element at the **`Representation`** level with a `@schemeIdUri` attribute equal to `http://dashif.org/guidelines/watermarking_wmpaceinfo` and `@value` attribute equal to the pointer to the sidecar file. The pointer is relative.

The following is an example of a DASH ingest manifest, the watermarking signalling is highlighted in red. In this example, the absolute path for the sidecar file for the first representation is equal to https://dash.edgesuite.net/dash264/TestCases/1a/ElephantsDream_H264BPL30_0100.264.dash_wm_pace_info.

NOTE: For every Variant, the information within the sidecar file is only different for the `variant` value. As there is only one sidecar file per `Representation`, the “variant” value is not relevant.

NOTE: This example also includes the signalling defined in clause 5.3.4 for defining the paths to segments from the Variants A and B. In this case, the **`EssentialProperty`** elements are added in the **`Representation`** element for modifying the **`BaseURL`** element value that is also in the **`Representation`** element. The **`BaseURL`** element value at the **`MDP`** level is not modified.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 DASH-MPD.xsd"
  type="static"
  mediaPresentationDuration="PT654S"
  minBufferTime="PT4S"
```

```
  <BaseURL>https://dash.edgesuite.net/dash264/TestCases/1a/</BaseURL>
  …
  <AdaptationSet mimeType="video/mp4" codecs="avc1.42401E" subsegmentAlignment="true"
subsegmentStartsWithSAP="1" contentType='video' maxWidth="480" maxHeight="360"
maxFrameRate="24" par="4:3">
    <Representation id="2" bandwidth="150000" width="480" height="360" frameRate="24">
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a"
value="a/ElephantsDream_H264BPL30_0100.264.dash"/>
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#b"
value="b/ElephantsDream_H264BPL30_0100.264.dash"/>
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_wmpaceinfo"
value="ElephantsDream_H264BPL30_0100.264.dash_wm_pace_info"/>
      <BaseURL>ElephantsDream_H264BPL30_0100.264.dash</BaseURL>
      <SegmentBase indexRange="984-11244">
        <Initialization range="0-983"/>
      </SegmentBase>
    </Representation>
    <Representation id="3" bandwidth="250000" width="480" height="360" frameRate="24">
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#a"
value="a/ElephantsDream_H264BPL30_0175.264.dash"/>
      <EssentialProperty
schemeIdUri="http://dashif.org/guidelines/watermarking_variant#b"
value="b/ElephantsDream_ H264BPL30_0175.264.dash"/>
      <EssentialProperty
schemeIdUri="https://dashif.org/guidelines/watermarking_wmpaceinfo"
value="ElephantsDream_H264BPL30_0175.264.dash_wm_pace_info"/>
      <BaseURL>ElephantsDream_H264BPL30_0175.264.dash</BaseURL>
      <SegmentBase indexRange="984-11245">
        <Initialization range="0-983"/>
      </SegmentBase>
    </Representation>
  …
  </AdaptationSet>
</MPD>
```

HLS ingest playlists shall include in the media playlist a custom tag specifying the pointer to the sidecar file. The pointer is relative. The tag is `#EXT-X-WMPACEINFO`. In the media playlist for each Variant (A, B, C …), the sidecar file referenced by the `#EXT-X-WMPACEINFO` tag is the same as the `variant` value shall not be considered.

The following is an example of a HLS media playlist, the watermarking signalling is highlighted in red. Note that the master playlist remain unmodified.

```
#EXTM3U
#EXT-X-TARGETDURATION:8
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-WMPACEINFO:"main_wm_pace_info"
#EXT-X-MAP:URI="main.mp4",BYTERANGE="1118@0"
#EXTINF:7.98333,
#EXT-X-BYTERANGE:1700094@1118
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1789481@1701212
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1777588@3490693
a/main.mp4
#EXTINF:8.00000,
#EXT-X-BYTERANGE:1752144@5268281
a/main.mp4
#EXTINF:7.26667,
#EXT-X-BYTERANGE:1563219@7020425
a/main.mp4
```

5.3.5. Encoding Recommendations

This clause contains recommendation when encoding content. The goal is to facilitate the creation and management of A and B Variants in the delivery chain.

When segments are requested as byteranges in a file, the A and B Variants shall be aligned as byteranges must be the same in both Variants, the player receives only one DASH manifest or HLS playlist and will get byterange values from one `sidx` box only. The added data shall not break the player. How this is achieved in out of the scope of this document (as an example, bit stuffing in the encoder is an option).

NOTE: This solution does not allow creating aligned segment when content is delivered with HLS in the form of MPEG-2 TS segments encrypted with AES sample encryption.

NOTE: An alternative solution is to not use segments requested as byteranges, but to use discrete files, in this case, there is no need to align Variant A and B of the same segment.

In case of ad insertion, the break may happen at any time. As every segment carries watermarking information allowing to perform the detection, there shall not be segments carrying conflicting data. While some techniques may recover from this mix of data, it will, in all cases, impact the length of content needed for retrieving the `WMID`.

Assuming that a replacement period is defined, from the device perspective, the following consumption modes are possible

- The device consumes ads from an alternative edge for the full duration of the ad break
- The device consumes ads from an alternative edge for a duration shorter than the replacement period
- The device consumes the original content as no alternative ad is proposed

All these devices are therefore consuming content differently during the ad break. The watermarking shall remain consistent between all these options.

The encoder, as part of its configuration, has a segment duration defined as written in [6]. Among other things that allows the encoder to assign every segment to a `pos` value in `WMPaceInfo`. In case of an ad break, the last segment before the ad break may be shorter than the previous ones. The encoder shall use the early signalization in the stream for detecting this configuration (SCTE signaling for example).

NOTE: Even if this segment is shorter, it may still be large enough for the packager for creating an independent segment.

The encoder shall apply no watermarking on every frame that is in this shorter segment. In this case, the encoder shall forward to the packager `WMPaceInfo` that contains `iswm` equal to false. If this segment is merged with the previous segment, `WMPaceInfo` shall then be discarded by the packager as it only uses the `WMPaceInfo` from the previous segment.

Devices receiving an ad for replacement shall receive it from a different edge server that does not enforce watermarking. Such edge server will then gracefully ignore the WM token.

Some devices may receive the original content if no ad can be found for replacement. One consequence is that these devices receive content that is meant to be watermarked following the rules of this document. The encoder shall apply no watermarking on all segment part of the ad break. In this case, the encoder shall forward to the packager `WMPaceInfo` that contains `iswm` equal to false.

5.3.6. Packaging Recommendations

This clause contains recommendation when packaging content. The goal is to facilitate the creation and management of A and B Variants in the delivery chain.

The minimum segment duration should take into account the embedding capabilities of the WM technology in order to ensure that a segment contains only information for A or B Variant. A segment carrying only one bit of information (Variant A or B) allows to match a segment to a bit value in the WM pattern.

Segment file naming with template based on segment $number may be used. The segments may also be defined based on $time. In the second case, an agreed fragment duration and time resolution shall be used. This is then provided to the edge as the `segduration` field in the watermark-token (see clause 5.4). The targeted segment duration and time resolution shall be used to define the value of `segduration` parameter.

Packager aggregates received parts of content. It builds a segment beginning with the part of content with firstpart=true and then aggregates nbpart for creating a segment until the targeted length has been reached. It shall begin creating a new segment if a part of content with firstpart=true is received before reaching the targeted length.

The transformation of ingest manifest into egress manifests requires the following actions:

- All "watermarking_wmpaceinfo" **EssentialProperty** elements in DASH manifests and EXT-X-WMPACEINFO tags in HLS playlists shall be removed from the egress manifests.
- All "watermarking_variant" **EssentialProperty** elements in DASH manifest shall be removed from the egress manifests
- A and B HLS media playlists of a given rendition in HLS shall be merged into a single, neutral version of it (without a/ or b/ sub-paths).

5.4. WM Token

5.4.1. General Principles

The WM token provides (either directly or indirectly) a WM pattern which is unique (for example per streaming session or per user). This pattern allows the selection of a specific sequence of A/B Variants. The token is obtained prior to any request for content, as it is needed in every Variant request with the edge server and will be passed through by the player.

Two tokenisation schemes are defined in this document. The first, named direct, embeds the WM pattern in the token and can be opened and interpreted by a edge server irrespective of the underlying WM technology and provider. The second, named indirect, requires integration of a WM technology provider's edge sequencing software at the edge server.

The following are requirements on the generated WM token:

- The token shall be JWT token, the basic structural requirements are defined in [4] and [5].
- The token shall include either a WM pattern or data for deriving the WM pattern.
- The token shall be signed as described in [5]. It shall use RS256 signature, pem key. The alg header parameter defined in [5] shall be present.
- Implementations shall process claims listed in [4] clause 4.1 when they are present. Among these claims, exp shall be present.
- The typ header parameter ([4] clause 5.1) should not be present.

In addition, the following claims are defined:

Claim	Type	Claim Name	Use	Explanation
wmver	uint	WM Token Schema Version	M	Version of the WM Token. This document describes version 1.
wmvnd	string	WM Vendor	M	WM vendor identification.
wmidtyp	uint	WMID Type	M	When set to 0, this token is used in a direct case. When set to 1 this token is used in an indirect case.
wmidfmt	string	WMID Format	M	Provides the representation format used for wmid. Possible values are "base64", "hexascii", "uint" and "ab" [i.1]. base64 support is mandatory, others are optional. NOTE: "ab" specifies the pattern as sequence of "A" and "B" characters. As an example, wmid="ABBAB", the pattern is interpreted from left to right, similar to the way an index into an array would work, an index of 3 returns "A".
wmpatlen	uint	WM Pattern Length	M	Provides the length of WM pattern extracted or derived from wmid.
wmid	string	WMID	M	When wmidtyp=0, it contains the WM pattern. When wmidtyp=1, it is used as input to derive the WM pattern. The derivation algorithm is not defined in this document.
segduration	uint	Nominal duration of a segment	O	When WMPaceInfo is not available, this may allow the edge to define the index to be considered in the WM pattern. If WMPaceInfo is present, the value of this claim shall be ignored.
wmidalg	string	WMID Decryption Algorithm	O	Provides the identification of the algorithm to use to decrypt wmid when encrypted. Possible values are "aes-128-cbc" and "aes-256-cbc".
wmidivlen	uint	WMID IV Length	O	Provides the length of the IV to use to decrypt wmid when encrypted.

wmidivhex	string	WMID IV	O	Provides the IV to use to decrypt wmid when encrypted. It is an hexascii string.
wmidpid	string	WMID Password ID	O	Provides the ID of the password to use for deriving a key to decrypt wmid when encrypted.
wmidpalg	string	WMID Password Algorithm	O	Provides the method for deriving from the password a key to decrypt wmid when encrypted. Possible value is "sha256".
wmidkeyver	uint	WMID Key Version	O	When wmidtyp=0, this shall not be used. When wmidtyp=1 it identifies the key to use for derivating the WM pattern.
wmopid	uint	WM Operator ID	O	When wmidtyp=0, this shall not be used. When wmidtyp=1 it may provide additional data for the derivation algorithm that generates the WM pattern.
M=mandatory, O=optional				

When wmidtyp=0 (direct case), it is recommended to encrypt wmid.

NOTE: for the use of segduration, it is assumed that linear services have a fix segment duration, except at start and end of ad insertions. For example, a fixed 2000 or 2002 msec duration of segments may be assumed, with an occasional exception that may interrupt the cadence. IT is therefore assumed that the watermark technology vendor supports these occasional breaks of the cadence, including at the time of watermark detection.

5.4.2. Direct Case: WM Pattern in the Token

The following example is an excerpt from the WMT payload for the direct case. Note that none of the claims indirectly pointing to the WM pattern are included in the WMT payload. In this example, wmid is encrypted.

```
{
    …
    "wmver":     1,
    "wmidtyp":   0,
    "wmidfmt":   "base64",
    "wmpatlen":  128,
    "wmid":      " 5hOdS05QcLFVSyjlZnF9mDGR1ipqw949MqYfanFIyMI= ",
    "wmidalg":   "aes-128-cbc",
    "wmidivlen": 16,
    "wmidivhex": " a45890072f06aebaa4786fb540ab707a ",
    "wmidpid":   "decryptpw_2017-06-28",
    "wmidpalg":  "sha256"
    …
}
```

The flow of the operations when retrieving the WM pattern for the above example is shown in Figure 5. Note that each line annotated with the "select" comment, assumes multiple predefined choices. Note also that the claims listed below constitute the minimal set of claims necessary to produce a valid WM pattern when wmid is encrypted.

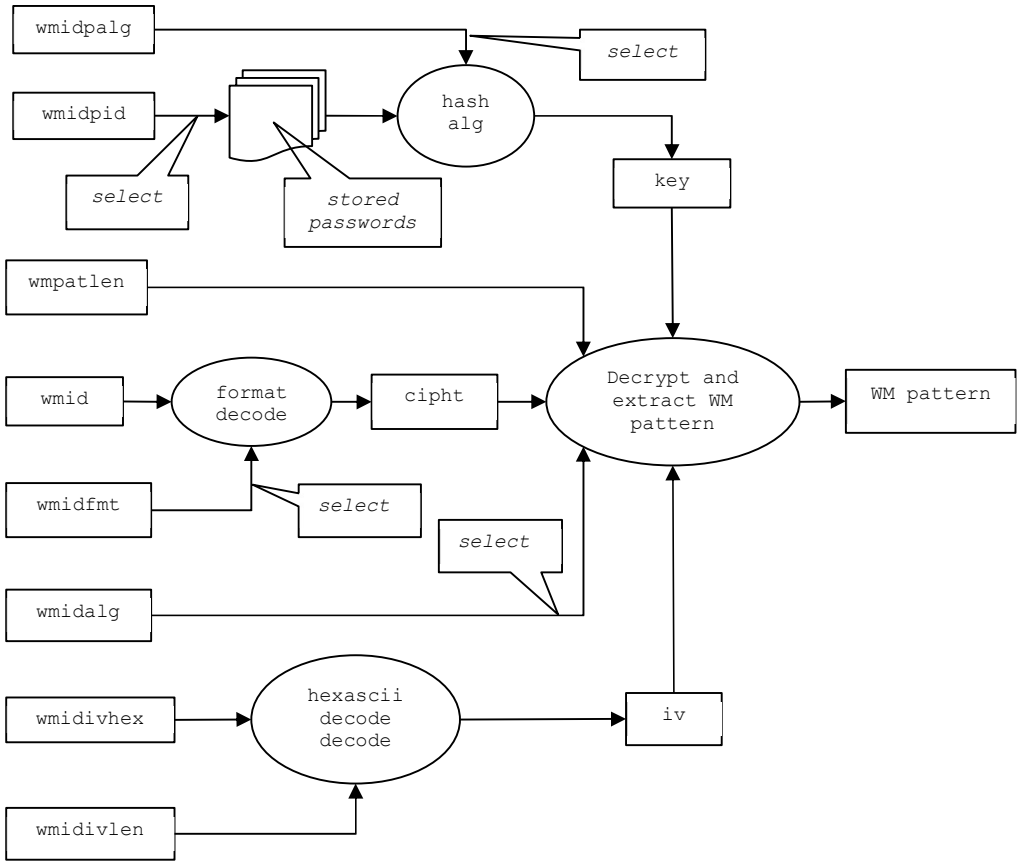


Figure 5: Example of flow for retrieving the WM pattern in the direct case.

5.4.3. Indirect Case: WM Pattern Derived from the Token

The following example is an excerpt from the WMT payload for the indirect WM pattern. Note that none of the claims directly pointing to the WM pattern are included in the WMT payload.

```
{
    …
    "wmver":      1,
    "wmvnd":      "<vendor>",
    "wmidfmt":    "uint",
    "wmidtyp":    1,
    "wmidkeyver": 2,
    "wmpatlen":   512,
    "wmid":       12345678,
    "wmopid":     25
    …
}
```

The flow of the operations when deriving the WM pattern from the provided parameters for the above example is shown in Figure 6. Note that there is a vendor specific core (selected by `wmvnd`). It is recommended that, performance-wise and software-stack-wise, it is comparable with the direct case. In other words, the vendors specific core should be based on the crypto operations which are used in the case `wmidtyp=0`, and, its performance should be equivalent. For example, the direct case relies upon one hash and one decryption operation when `wmid` is encrypted, the vendor specific core should be consisting of the similar operations in order to preserve the quantity of operations similar between these two cases.

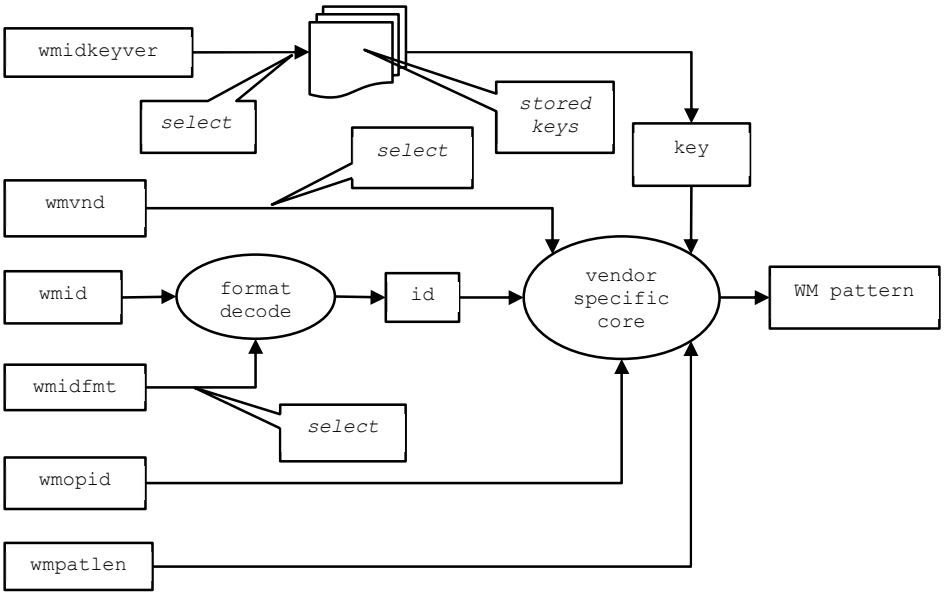


Figure 6: Example of flow for retrieving the WM pattern in the indirect case.

5.4.4. Using segduration

When `segduration` is provided in the watermark-token while there is no `WMPaceInfo`, the file `number` is considered a timestamp that needs to be converted to a pattern index with

Pattern index = int (`number`/`segduration`)% `wmpatlen`.

It is assumed a fixed 2000 or 2002 msec segment duration for all segments as explained in clause 5.3.6.

The following is an example of computation of the pattern index where

- Content is packaged with 100 nanoseconds timestamp unit.
- `segduration` is equal to 20000000 (the segment duration valule corresponds to an expected 2000 msec with the 100 nanosecond timestamp representation).
- `wmpatlen` is equal to 1024.

For a request such as https://premium.cdn.edge/wmt:[token]/sports2/720p/fragment-30000000000.m4s, the following shall be calculated by the edge

- Calculate from `number` in the filename: (30000000000 / 20000000) = 1500
- Do modulo of the resulting value based on 1024 pattern length = 476
- Use A/B decision at pattern index 476 in the watermark pattern
- Rewrite the request to retrieve data from the origin (or from the cache)

5.5. Content Playback

5.5.1. Introduction

The flow for content playback is shown in the following figures and clauses. The origin received content as explained in clause 5.3. It has access to the A/B Variants and the data from the `WMPaceInfo`.

Content playback is divided in three actions:

- Acquiring the WM token, the DASH manifest or the HLS playlists
- Acquiring the initialisation segment
- Acquiring media segments

While the first action is common to all type of content, the other ones have variations depending on the packaging and delivery mode of the content. Variation is, for example on the difference between content delivered as byterange and content delivered as discrete segments. Another possible variation appears when HLS low latency is used for the chunks requested at the edge of live.

The following goes through the different actions by providing the expected workflows.

5.5.2. WM Token, DASH Manifest and HLS Playlists Acquisition

The device acquires the WM token in an implementation specific manner. It may be retrieved directly from a WM token server or it may be provided in a response from another server as part of other data required for playing back content.

The WM token may be added as part of the virtual path of the requested object, as a query string attribute or as part of the HTTP header when the device requests content to the edge server. It is recommended to use the virtual path.

The WM token may be added by the device for requesting DASH manifest and HLS playlists. While these objects are not watermarked (the pattern in the name allows the edge server to know this), the edge server may validate or not the token and refuse to serve these objects if the token is not valid. The CDN edge may also gracefully ignore the token in this case, as the names of the requested files do not match the pattern for watermarked content. The origin server cleans the served objects, removing any property related to location of objects. The manifest and playlist shall be neutral as explained in clause 5.3.6. This is summarized in Figure 7.

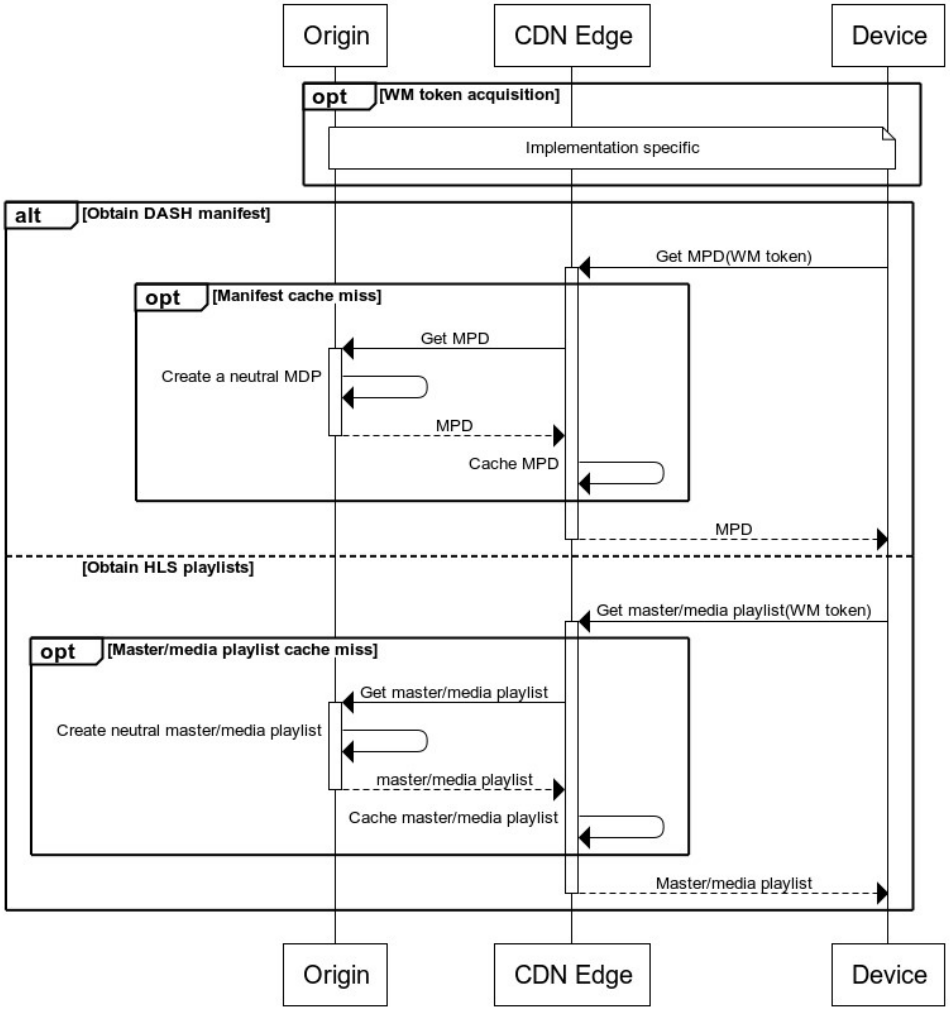


Figure 7: Token, DASH manifest and HLS playlist acquisition.

5.5.3. Initialisation Segment Acquisition

When content is delivered as byteranges, as the initialisation segment is within the file, the token shall be added in the request as the requested file has a name that matches the pattern for watermarked content. The edge server will then apply the exact same logic it applies for a media segment, it retrieves the sidecar file and extracts the `WMPaceInfo` for the first part of the track that contains the initialisation segment (as defined in clause 5.5.4). It can then deliver the initilisation segment to the device. The flow is shown in Figure 9 where the initialisation segment is treated as a media segment.

NOTE: The token is evaluated as the edge server cannot make a difference between the initialisation segment and a media segment.

In case content is delivered as discrete segments, the name of the initialisation segment does not match the pattern for watermarked content as written in clause 5.3.4.2. The WM token may be added by the device for requesting the initialisation segment. The edge server may validate it or not and may refuse to serve these objects if it is not valid. The edge server may also gracefully ignore it. The flow is shown in Figure 7.

5.5.4. Media Segments Acquisition

5.5.4.1. General Requirements

For the media segments, a token shall be attached to the HTTP requests. If not present, the edge server shall reject the request and shall not deliver the segment.

For each device request for `/pathname/filename`, the edge server shall send a request to the origin server for `/pathname/WMPaceInfo/filename` for retrieving the `WMPaceInfo` data associate to this object. The `Content-Type` for this object is `application/json`. The origin server response shall carry a sidecar file that includes the `WMPaceInfo` data in the payload:

- For a discrete segment request (`filename` is a segment file), the response payload shall contain a single `WMPaceInfo` object in a sidecar file as defined in clause 5.3.3.6 for discrete segments. If `WMPaceInfo` was delivered to the origin server as part of the HTTP header, SEI message, ISOBMFF box, TS adaptation field or a sidecar file per track, it shall extract the data to make it available to the edge server as a HTTP payload in the form of a sidecar file that contain a single `WMPaceInfo` object when the information is requested by the edge server.
- For a byterange request (`filename` is a track file with a byte range), the response payload shall contain the sidecar file (a list of `WMPaceInfo` objects as defined in clause 5.3.3.6 for byterange segments). The origin server shall not extract data from the file and only provide the sidecar file to the edge server.

NOTE: As the data is always delivered as sidecar file between the origin and the edge, it is recommended that the origin always store it in sidecar file.

Any direct request from a device with `/pathname/WMPaceInfo/filename` shall receive a 403 error code.

Following the example given in clause 5.3.1, the following gives examples of content flows as ingest to the origin and egress of the origin to the edge:

	Live content	**VOD content**
Ingest of the origin	No sidecar file, data is delivered as part of HTTP headers, SEI messages, ISOBMFF boxes or TS adaptation fields.	One sidecar file per Representation either with discrete segments or byteranges.
Egress of the origin (payload of the HTTP response)	One sidecar file per segment (note the special case of HLS low latency with byterange where multiple chunks are be linked to the same sidecar file, see clause 5.5.4.3).	For discrete segments, one sidecar file per segment. For byterange, one sidecar file per Representation.

There are then three entry points on the origin server:

- `WMPaceInfo`: /pathname/WMPaceInfo/filename
- Variant A: /pathname/`subPath`/filename
- Variant B: /pathname/`subPath`/filename

Where `subPath` (part of `variantSubPaths`) is as defined in clause 5.3.3.6.

NOTE: Adding Variants creates additional entry points.

The connection between the origin server and the edge server shall be restricted to legitimate requests. How this is achieved is out of the scope of this specification.

NOTE: A static secret (a shared key), dynamic signatures or access lists (based on IP addresses) are examples of tools for restricting the access.

One or several Variants may become unavailable on the origin for any reason, such as a lost connection with the encoder for these encoding pipelines. Devices request all Variants, such situation will result in intermittent black screens when requesting the affected Variants. The origin server shall deliver any available Variant.

NOTE: This is breaking the watermarking detection. The period of time when such contingency measure is applied is not to be used for detection. How the end-to-end system is synchronized is out of the scope of this document. As an example, the origin server can raise an alarm.

5.5.4.2. Discrete Files

For the media segments delivered as discrete files, the flow is shown in Figure 8. The edge server shall validate the WM token (that can include checking signed data or decrypting some claims) which is attached to the requests and extracts the WM pattern. The edge server delivers the A or B Variant of a segment based on the WM pattern contained in the token. In order to know which position in the WM pattern it has to consider, it first makes a HTTP GET request to the origin in order to retrieve the `WMPaceInfo` data. This is done with a GET request using the path `/pathname/WMPaceInfo/filename`. The origin server provides the `WMPaceInfo` from the Variant A in the payload of the response.

NOTE: `WMPaceInfo` contains almost the exact same information for all Variants (except the Variant identification), hence only one request is needed for retrieveing the information needed for the decision.

Once, the data in `WMPaceInfo` is interpreted in conjunction with the WM pattern, the edge server can deliver the right Variant corresponding to the position in the WM pattern that matches the value of `pos` in `WMPaceInfo`.

As an alternative, the edge may use segduration from the token for defining the position on the WM pattern. Refer to clause 5.4.4 for more details.

The edge server caches the Variants of a given segment with different cache keys and it should prevent the cache keys to be revealed through debug headers.

The origin server shall strip out from video segments client responses all `WMPaceInfo` headers that would have been generated by the upstream transcoder. It shall also zero-pad WMPaceInfo when carried as SEI messages, TS adaptation fields or ISOBMFF boxes.

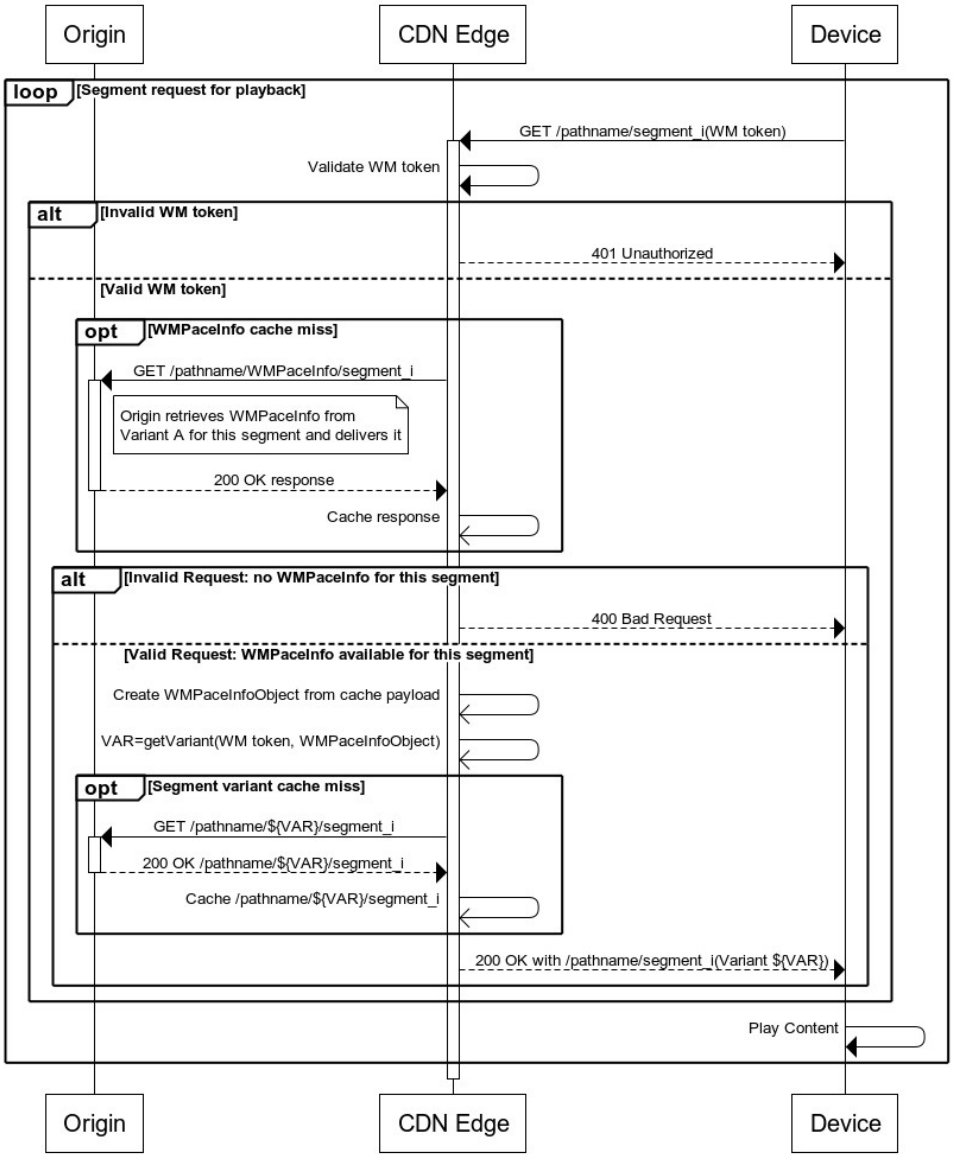


Figure 8: Media segment, as discrete file, acquisition.

5.5.4.3. Byterange

For the media segments delivered as byteranges, the flow is shown in Figure 9. The edge server validates the WM token (that can include checking signed data or decrypting some claims) which is attached to the requests and extracts the WM pattern.

The edge server delivers the A or B Variant of a segment based on the WM pattern contained in the token. In order to know which position in the WM pattern it has to consider, it needs to retrieve the sidecar file associated to this track. It first makes a HTTP GET request to the origin in order to retrieve the sidecar file.

From this sidecar file, the edge server can extract the `WMPaceInfo` corresponding the requested byterange. If the requested byterange does not have a corresponding `WMPaceInfo`, the edge server shall not serve content to the device.

Content delivered with HLS using the `EXT-X-PART` tag are byterange requests within a discrete segment. When the edge server receives the request for this partial segment, it will request `WMPAceInfo` to the origin server and will receive a sidecar file with only one `WMPAceInfo`. This allows the edge server to know that it shall not enforce byterange validation for these requests.

NOTE: Only byteranges overlapping a valid byterange are problematic, requests for byteranges included in valid values are not breaking the WM pattern that is created by the A/B Variants.

Once, the data in `WMPaceInfo` is interpreted in conjunction with the WM pattern, the edge server can deliver the right Variant corresponding to the position in the WM pattern that matches the value of `pos` in `WMPaceInfo`.

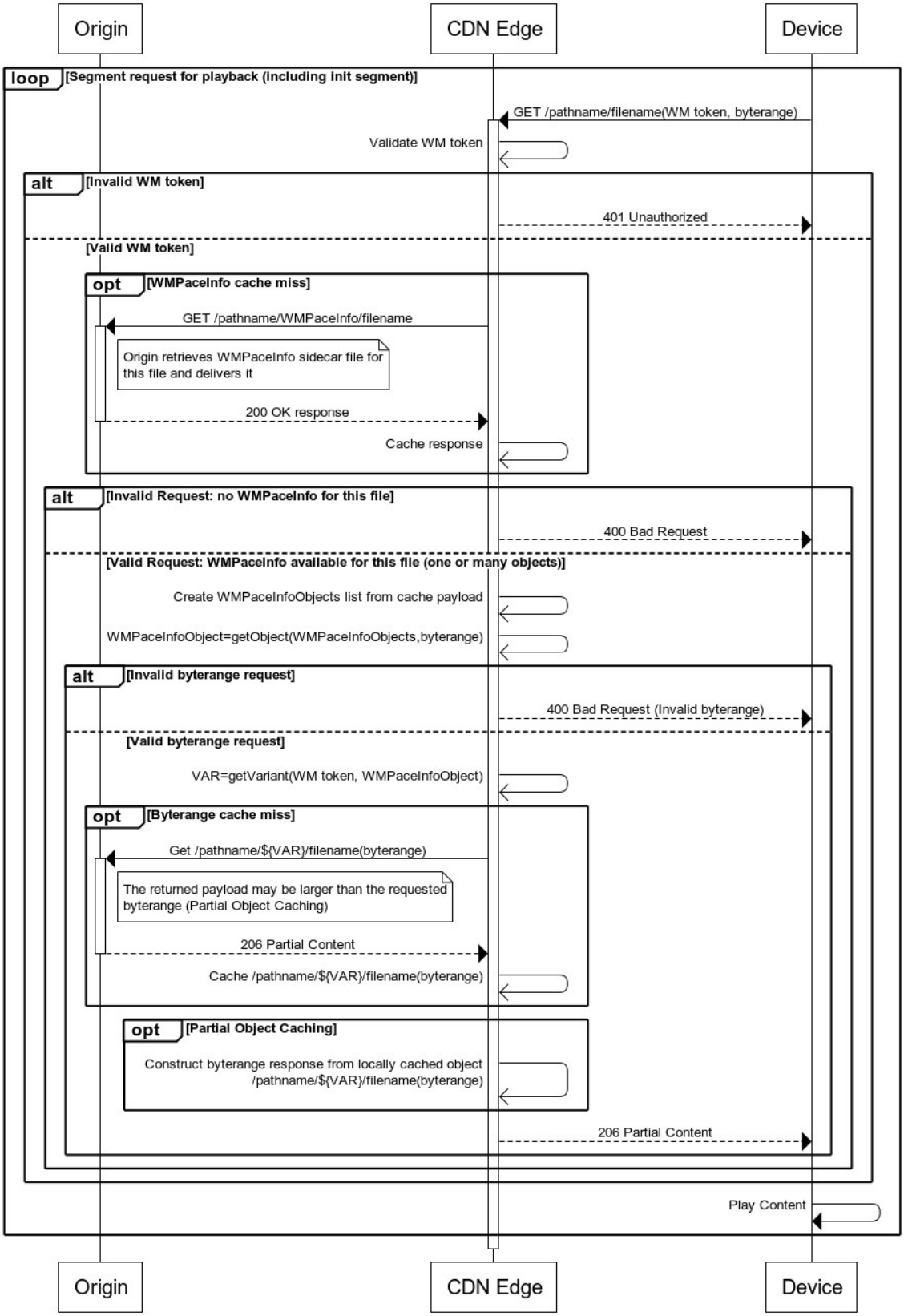


Figure 9: Media segment, as byterange, acquisition.

5.5.5. Toggling Off Watermarking Enforcement

Watermarking enforcement is not always necessary and there are cases where this may be needed either episodically or only for a limited period of time. Examples of such use cases are:

- A live service of sport events where some premium events only require watermarking enforcement. Other moments of the day do not require it. In this case, content is always watermarked but the edge server is enforcing it only during limited periods of time.
- A premium asset of content just added to a VOD service requires watermarking enforcement during the first two weeks of release and then it is not needed anymore. Content has been watermarked when released but after this enforcement period, the edge server is not applying enforcement anymore.

Toggling off watermarking enforcement allows avoiding the duplication of content in the system when not needed. It is important to note that content is watermarked, as it is not possible in such cases to delete existing watermarked content. The origin server still has all Variants but it does not propagate them in the system. The flow for the preparation of content is not modified. Content is still delivered to the origin server with multiple Variants. Segments have `WMPaceInfo` associated information. Refer to clause 5.3 for additional information.

Once received at the origin server, `WMPaceInfo` carried as part of the payload to the edge server is modified. More specifically, toggling off the watermarking enforcement is announced to the edge server by setting `iswm` in `WMPaceInfo` to `false`.

NOTE: This assumes that the origin server is receiving watermarking enforcement on/off information from a content management system using mechanisms out of the scope of this specification.

On its side, the device still acquires a WM token as the device consumes content without knowing whether enforcement of watermarking is enabled or not. If fact, the device may consume watermarked or non-watermarked content while the WM token is valid. This shall be fully transparent to devices.

As playback, the edge server receives from a device a request for a segment and retrieves `WMPaceInfo` from the origin. As `iswm` is equal to `false`, the `getVariant` function (as shown in Figure 8 and Figure 9) shall output “A”, meaning that the edge always consume segment on the A entry point regardless of the values in WM pattern.

Figure 3

```
Participant Live ABR Transcoder
Participant Packager
Participant Origin

# STEP 1: Ingest from the transcoder to the packager
# For instance, the segmentation is 1s long
Live ABR Transcoder-> Packager: Ingest manifest
Live ABR Transcoder-> Packager: Ingest segments Variant A\n (w/ WMPaceInfo)
Live ABR Transcoder-> Packager: Ingest segments Variant B\n (w/ WMPaceInfo)

# STEP 2: Ingest from the Packager to the Origin (e.g. 2S long segments)
# The Packager has to aggregate several DASH segments to produce the distributed segment
Packager-> Origin: Egress manifest
Packager-> Origin: Egress segments Variant A\n (w/ WMPaceInfo)
Packager-> Origin: Egress segments Variant B\n (w/ WMPaceInfo)
```

Figure 7

```
Participant Origin
Participant CDN Edge
Participant Device

# STEP 1: Acquire a WM token
opt WM token acquisition
    note over Origin,Device: Implementation specific
end

# STEP 2 : Get the DASH manifest or HLS playlist for the viewing session
alt Obtain DASH manifest
    Device->+CDN Edge: Get MPD(WM token)
    opt Manifest cache miss
        CDN Edge->+Origin: Get MPD
        Origin->Origin: Create a neutral MDP
        Origin-->-CDN Edge: MPD
        CDN Edge->CDN Edge: Cache MPD
    end
    CDN Edge-->-Device: MPD
else Obtain HLS playlists
    Device->+CDN Edge: Get master/media playlist(WM token)
    opt Master/media playlist cache miss
        CDN Edge->+Origin: Get master/media playlist
        Origin->Origin: Create neutral master/media playlist
        Origin-->-CDN Edge: master/media playlist
        CDN Edge->CDN Edge: Cache master/media playlist
    end
    CDN Edge-->-Device: Master/media playlist
end
```

Figure 8

```
Participant Origin
Participant CDN Edge
Participant Device

loop Segment request for playback
    Device->+CDN Edge: GET /pathname/segment_i(WM token)
    CDN Edge->CDN Edge: Validate WM token
    alt Invalid WM token
        CDN Edge-->Device: 401 Unauthorized
    else Valid WM token
        opt WMPaceInfo cache miss
            CDN Edge->+Origin: GET /pathname/WMPaceInfo/segment_i
            note right of Origin
                Origin retrieves WMPaceInfo from
                Variant A for this segment and delivers it
            end note
            Origin-->-CDN Edge: 200 OK response
            CDN Edge ->> CDN Edge: Cache response
        end
        alt Invalid Request: no WMPaceInfo for this segment
            CDN Edge-->Device: 400 Bad Request
        else Valid Request: WMPaceInfo available for this segment
            CDN Edge ->> CDN Edge: Create WMPaceInfoObject from cache payload
```

```
            CDN Edge ->> CDN Edge: VAR=getVariant(WM token, WMPaceInfoObject)
            opt Segment variant cache miss
                CDN Edge->+Origin: GET /pathname/${VAR}/segment_i
                Origin-->-CDN Edge: 200 OK /pathname/${VAR}/segment_i
                CDN Edge ->> CDN Edge: Cache /pathname/${VAR}/segment_i
            end
            CDN Edge-->Device: 200 OK with /pathname/segment_i(Variant ${VAR})
        end
    end
    Device->Device: Play Content
end
```

Figure 9

```
Participant Origin
Participant CDN Edge
Participant Device

loop Segment request for playback (including init segment)
    Device->+CDN Edge: GET /pathname/filename(WM token, byterange)
    CDN Edge->>CDN Edge: Validate WM token
    alt Invalid WM token
        CDN Edge-->Device: 401 Unauthorized
    else Valid WM token
        opt WMPaceInfo cache miss
            CDN Edge->+Origin: GET /pathname/WMPaceInfo/filename
            note right of Origin
                Origin retrieves WMPaceInfo sidecar file for
                this file and delivers it
            end note
            Origin-->-CDN Edge: 200 OK response
            CDN Edge ->> CDN Edge: Cache response
        end
        alt Invalid Request: no WMPaceInfo for this file
            CDN Edge-->Device: 400 Bad Request
        else Valid Request: WMPaceInfo available for this file (one or many objects)
            CDN Edge ->> CDN Edge: Create WMPaceInfoObjects list from cache payload
            CDN Edge ->> CDN Edge: WMPaceInfoObject=getObject(WMPaceInfoObjects,byterange)
            alt Invalid byterange request
                CDN Edge-->Device: 400 Bad Request (Invalid byterange)
            else Valid byterange request
                CDN Edge ->> CDN Edge: VAR=getVariant(WM token, WMPaceInfoObject)
                opt Byterange cache miss
                    CDN Edge->+Origin: Get /pathname/${VAR}/filename(byterange)
                    note right of Origin
                        The returned payload may be larger than the requested
                        byterange (Partial Object Caching)
                    end note
                    Origin-->-CDN Edge: 206 Partial Content
                    CDN Edge ->> CDN Edge: Cache /pathname/${VAR}/filename(byterange)
                end
                opt Partial Object Caching
                    CDN Edge->>CDN Edge: Construct byterange response from locally cached
object\n/pathname/${VAR}/filename(byterange)
                end
                CDN Edge-->Device: 206 Partial Content
            end
        end
    end
    Device->Device: Play Content
end
```