Edge Services: Watermarking

Overview for Watermarking Partners

Date	2020-07-21
Version	1.9

1 Executive Summary

Watermarking in the context of media delivery is the notion of adding PII to the content such that if the content is comprised (e.g. pirated, rebroadcasted) it can be traced back to the source of the leak. Watermarking does not address url redistribution or DRM, but instead provides the necessary forensic information in order to deter pirating at its source.

2 Architectural Overview

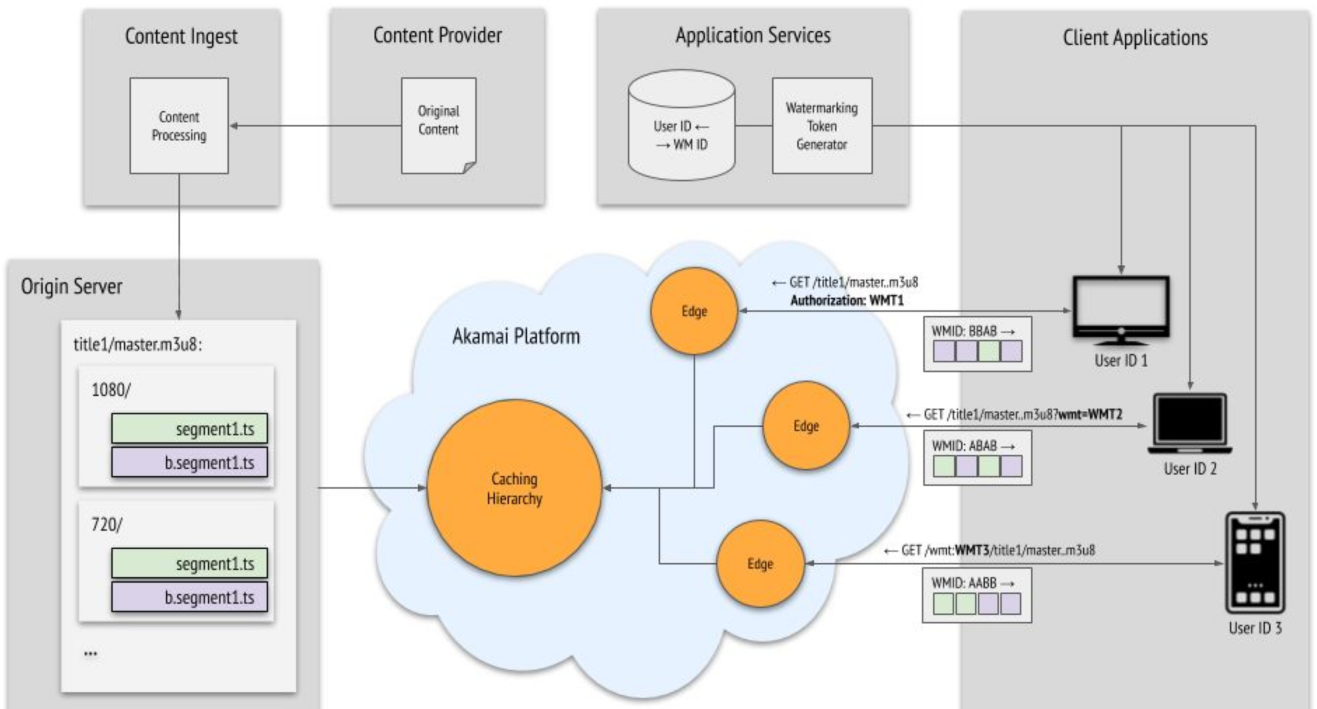
2.1 Overview

At a high level, this solution is an implementation of what is commonly referred to as **two-step watermarking**. Two-step watermarking consists of a content preparation stage (step 1) during ingest, and a personalization stage (step 2) during playback.

A core requirement of this type of watermarking is that content is delivered in a segmented fashion due to the unique property of being able to encode a pattern in the output stream when sourcing different variants of the content for each segment. For example, imagine a fictional scenario where two versions of the content are encoded, one with the letter “A” embossed in the image, and the other with the letter “B”. When the content is viewed, a unique sequence of A and B segments can be delivered forming a pattern unique to that individual viewer. Forensic analysis of that viewer’s stream can extract the AB pattern and the viewer can be identified.

2.2 Ecosystem

The following figure shows a high level end-to-end ecosystem that supports two-step watermarking:



A content provider delivers content to the OTT Provider in an unmarked state. The OTT Provider then prepares the content for watermarking, typically producing an HLS and DASH version with both an A and a B variant. The content is placed on an origin server where a CDN with Edge compute will then transparently deliver a unique sequence of A and B segments to individual end users.

The OTT Provider and/or client application have the responsibility of delivering to the Edge a Watermarking Token (WMT), which is a JWT/JWS that secures the delivery of that end user's watermarking pattern, to the CDN Edge. The CDN Edge then transparently delivers segments to the end user in accordance with the pattern found within the token.

In order to perform this Edge switching in a stateless fashion, the Akamai platform must deliver HLS media manifest files due to the need to enrich those manifests with information vital to the faithful execution of the watermarking pattern.

The following sections describe in more detail the various logical components in this sample ecosystem.

2.2.1 Content Processing

The Content Processing component, likely an encoder/packager, has the responsibility of producing watermarked content from unmarked, source material:

- Converts source material into A/B variants
- Stores A/B variants on origin server with differentiated A and B paths
- Writes manifests without any indication of A or B in the path

Example: given a presentation called "title1" with a master and media playlist and several TS segments, the directory structure on the origin server (when using filename-prefix AB naming) would look like:

<base_path>/title1/master.m3u8 ← contains relative paths to variants: 1080-media.m3u8, ...
<base_path>/title1/1080-media.m3u8 ← references relative paths to segments: 1080/segment1.ts, ...
<base_path>/title1/1080/**a**.segment_1.ts
<base_path>/title1/1080/**a**.segment_N.ts
<base_path>/title1/1080/**b**.segment_1.ts
<base_path>/title1/1080/**b**.segment_N.ts
<base_path>/title1/720-media.m3u8
<base_path>/title1/720/**a**.segment_N.ts
...

The filename prefix makes it more difficult to retrieve the content without the Edge AB switching function enabled, thereby adding an additional layer of security to the content (e.g. someone accidentally disables watermarking or creates a second delivery configuration using that origin).

However, as a convenience to the operator, the A variant can be undecorated, thereby making it almost like a "default" variant facilitating disabling of watermarking once the requirement has passed. This approach results in content that does not require Edge switching in order to deliver it. This would have the following directory structure on the origin server:

<base_path>/title1/master.m3u8 ← contains relative paths to variants: 1080-media.m3u8, ...
<base_path>/title1/1080-media.m3u8 ← references relative paths to segments: 1080/segment1.ts, ...
<base_path>/title1/1080/segment_1.ts ← this facilitates easy disabling of watermarking
<base_path>/title1/1080/segment_N.ts
<base_path>/title1/1080/**b**.segment_1.ts
<base_path>/title1/1080/**b**.segment_N.ts
<base_path>/title1/720-media.m3u8
<base_path>/title1/720/segment_N.ts
...

The above AB naming convention is known as the "filename prefix", with "use original as A" approach which is what is recommended as the best practice when preparing content.

Another approach Akamai supports is the "directory prefix" approach, where instead of prefixing the filename, you prefix the path right before the filename with a directory labeled either "a" or "b". For example, the above content would look like the following (highlighted to showcase the difference):

<base_path>/title1/master.m3u8 ← contains relative paths to variants: 1080-media.m3u8, ...
<base_path>/title1/1080-media.m3u8 ← references relative paths to segments: 1080/segment1.ts, ...
<base_path>/title1/1080/segment_1.ts ← this facilitates easy disabling of watermarking
<base_path>/title1/1080/segment_N.ts
<base_path>/title1/1080/**b/**segment_1.ts
<base_path>/title1/1080/**b/**segment_N.ts
<base_path>/title1/720-media.m3u8
<base_path>/title1/720/segment_N.ts
...

2.2.1.1 Separate Audio/Video Segments

If the content is prepared with separate audio and video segments, but with indistinguishable file extensions, then the CDN Edge will attempt to AB switch audio as well as video. To that end, the content preparation stage must prepare AB variants for the audio segments as well, even though the switching thereof will have no watermarking benefit.

If the audio segments can indeed be differentiated by file extension, then they must reside in the default location as AB switching logic will be bypassed. This is one of the primary benefits of the "use original as A" approach when preparing content for AB switching.

2.2.2 HLS Specific Requirements

For HLS, the recommended best practice is the "filename prefix", with "use original as A" because it is the most straight forward approach when preparing content. If "use original as A" is not used, then for ALL non switched

URLs, i.e. non-video segments, the objects must reside in a "default" location as specified in the various manifest files--if and only if they cannot be distinguished from switched URLs by file extension. For example, if audio and video is separated but both utilize the *.mp4 file extension, they will both be switched. This means the audio segments must have matching A and B versions because they will be switched just like for video.

2.2.3 DASH Specific Requirements

At this time, the only variant of DASH that is supported is SegmentTemplate using $Number$ expansion. Additionally, we require that the $Number$ macro be at the end of the filename, and prefixed with an underscore character. Additionally, we require any "init" URLs to be prefixed with additional naming of the form, "<something>_init.mp4", this helps the platform detect the file as an "init" request and therefore no AB switching should be done.

The following is a DASH MPD snippet with only some of the pertinent XML tags and attributes present:

```
<MPD>
  <Period>
    <AdaptationSet mimeType="video/mp4" contentType="video">
      <SegmentTemplate duration="120" timescale="30" startNumber="1"
                       media="$RepresentationID$/$RepresentationID$_$Number$.m4v"
                       initialization="$RepresentationID$/$RepresentationID$_init.m4v"/>
      ...
    </AdaptationSet>
    <AdaptationSet mimeType="audio/mp4" contentType="audio">
      <SegmentTemplate duration="192512" timescale="48000" startNumber="1"
                       media="$RepresentationID$/$RepresentationID$_$Number$.m4a"
                       initialization="$RepresentationID$/$RepresentationID$_init.m4a"/>
      ...
    </AdaptationSet>
  </Period>
</MPD>
```

2.2.4 Watermarking Token Generator

The Watermarking Token Generator has the responsibility of generating a unique pattern for end users, and packaging up that pattern in a compliant JWT/JWS object. Typically found hand-in-hand with a user database, part of the bookkeeping process is to associate tokens with user IDs for future lookup when illegally shared content has been discovered. Together, these components must:

- Issues watermarking tokens
- Store a mapping between end users and tokens
- Generating unique watermarking patterns and secure them by producing a signed JWT/JWS

2.2.5 CMS/Client Application

The CMS and/or Client Application must present the Akamai with the user's Watermarking Token (WMT), at a minimum, for each content request arriving at the CDN Edge. Several methods for supplying the token are supported within Akamai and described in the Token Location section later in this document.

2.2.6 Content Delivery Network

The CDN's responsibility is to faithfully execute the specified watermarking pattern while delivering a stream to an end user. The CDN:

- Denies all content requests without a token
- Verifies the signature of the token
- Denies all requests with an invalid token
- Unpacks tokens and maps requests to A/B versions of source material based on the watermarking pattern (WMID) within the token
- Will cycle the WMID as many times as needed to deliver the entire sequence of segmented content

2.3 Sequence Diagrams

In order to help solidify the solution, a few important data flows are presented to help clarify how watermarking tokens are generated and used.

2.3.1 Playback URL Generation: Token as a Virtual Directory

When a token is specified as a virtual directory in the path, and the manifest files use relative paths, a convenient side effect results whereby all child requests include the same parent path and therefore the token as well. For example, consider the following URL with a Watermarking Token (WMT) embedded as a virtual directory:

http://ott-provider.akamaized.net/**wmt:WMT**/title1/master.m3u8

Then, consider the master.m3u8 manifest to contain the following:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=2411189,RESOLUTION=1280x720,CODECS="avc1.640c28,mp4a.40.34"
Rendition0/playlist.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=1311189,RESOLUTION=854x480,CODECS="avc1.640c1f,mp4a.40.34"
Rendition1/playlist.m3u8

#EXT-X-STREAM-INF:BANDWIDTH=651189,RESOLUTION=426x240,CODECS="avc1.640c15,mp4a.40.34"
Rendition2/playlist.m3u8
```

All subsequent requests for any of the renditions, and segments contained therein will have the /wmt:WMT/ virtual directory in it:

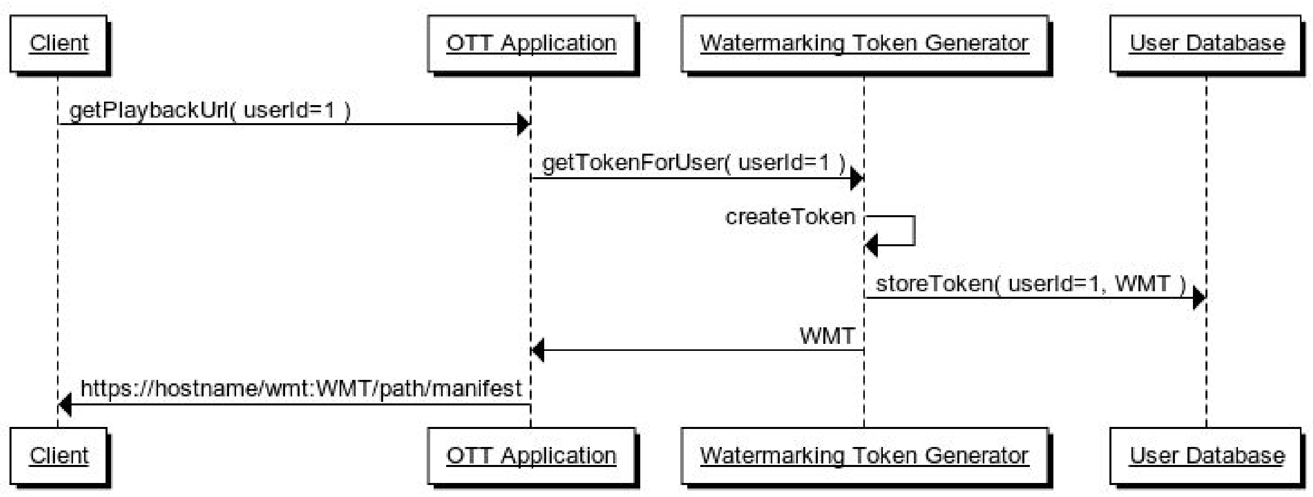
```
→ http://ott-provider.akamaized.net/wmt:WMT/title1/Rendition0/playlist.m3u8
```

```
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:2
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
```

```
#EXTINF:2.000000,
segment0.ts
#EXTINF:2.000000,
segment1.ts
```

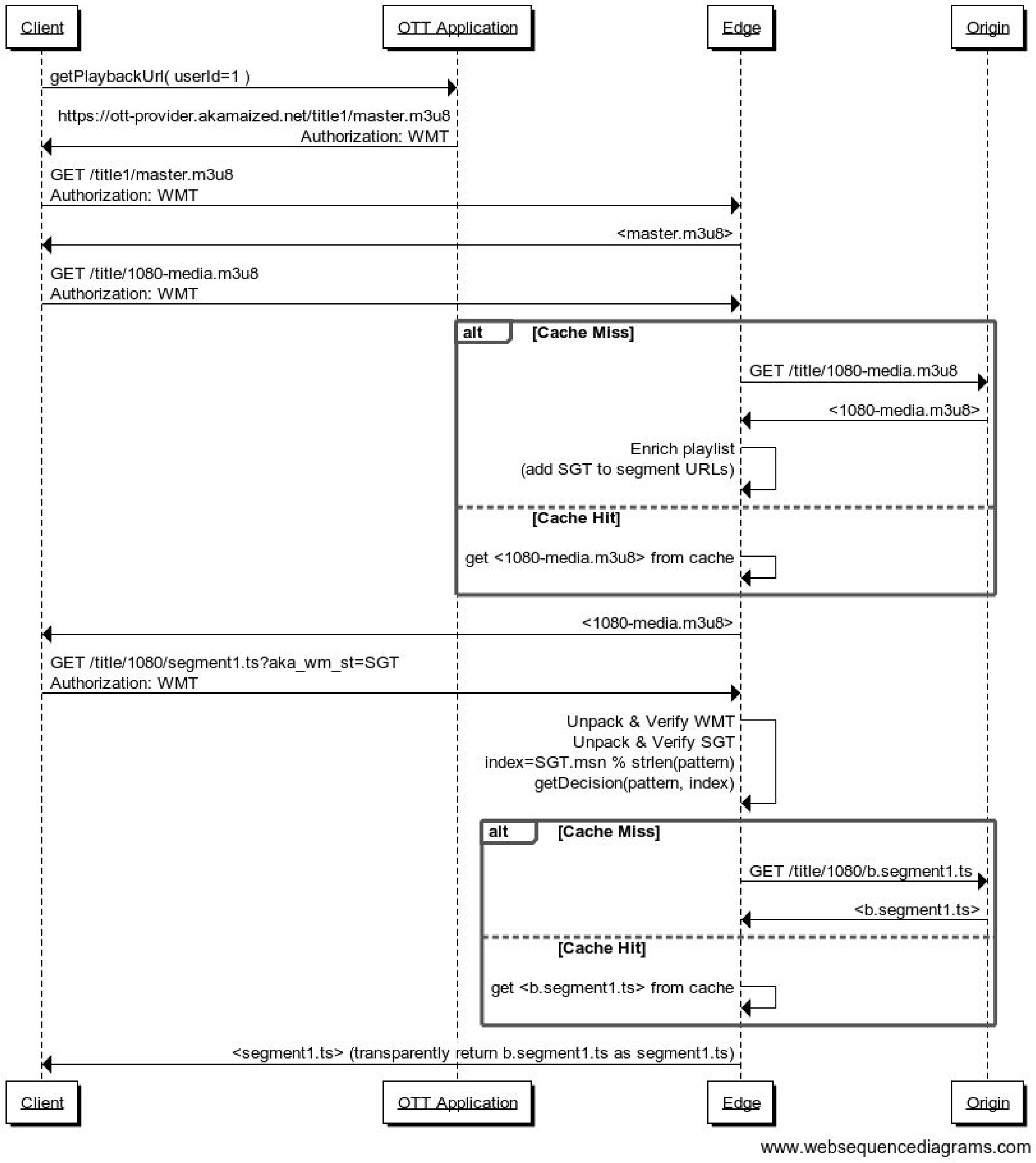
```
→ http://ott-provider.akamaized.net/wmt:WMT/title1/Rendition0/segment0.ts
```

Therefore, the benefit of a simple URL modification at the start of a playback session results in the token been included in each and every request from thereon in. The following sequence illustrates this workflow:


Client
OTT Application
Watermarking Token Generator
User Database
getPlaybackUrl(userId=1)
getTokenForUser(userId=1)
createToken
storeToken(userId=1, WMT)
WMT
https://hostname/wmt:WMT/path/manifest
Client
OTT Application
Watermarking Token Generator
User Database

2.3.2 Transparent AB Edge Switching

Part of the benefit of doing transparent Edge switching is the burden of generating a unique manifest for each and every stream is scaled out to the CDN Edge that can inherently handle the scale and load of this type of distributed decision logic. The following sequence diagram illustrates the segment retrieval workflow for Edge-based watermarking:


Client
OTT Application
Edge
Origin
getPlaybackUrl(userId=1)
https://ott-provider.akamaized.net/title1/master.m3u8
Authorization: WMT
GET /title1/master.m3u8
Authorization: WMT
<master.m3u8>
GET /title/1080-media.m3u8
Authorization: WMT
alt
[Cache Miss]
GET /title/1080-media.m3u8
<1080-media.m3u8>
Enrich playlist
(add SGT to segment URLs)
[Cache Hit]
get <1080-media.m3u8> from cache
<1080-media.m3u8>
GET /title/1080/segment1.ts?aka_wm_st=SGT
Authorization: WMT
Unpack & Verify WMT
Unpack & Verify SGT
index=SGT.msn % strlen(pattern)
getDecision(pattern, index)
alt
[Cache Miss]
GET /title/1080/b.segment1.ts
<b.segment1.ts>
[Cache Hit]
get <b.segment1.ts> from cache
<segment1.ts> (transparently return b.segment1.ts as segment1.ts)
Client
OTT Application
Edge
Origin
www.websequencediagrams.com

2.4 Watermarking Token

The watermarking token is sent by the client application for every segment or fragment request. The CDN both ensures a token is present and verifies the signature of the token in order to prevent tampering. The JSON Web Token (JWT) standard is used as a method for tokenizing data and providing security.

A sample implementation of a watermarking token can be seen via the following test page using my personal web server as a token generator:

http://acceleratedmedia.akamaized.net/demos/watermarking/

2.4.1 Token Format

The JSON Web Token (JWT) format consists of a header, payload, and signature concatenated with the period character:

Watermarking Token (WMT) = JWT = <header>.<payload>.<signature>

The header and payload are base64 encoded JSON objects with the signature being dependent on the algorithm used for signing. Let's look at an example:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJ1cm46ZmxhbWluaG9vcCIsImV4cCI6MTU1NDczOTE1OCwi
cGF0dGVybiI6IkFCQkFCQUFBQkJCQUJBQkJCQUFBQkJBQkJBIn0%3D.a3lPLdeCWN3aa2Z3z1IBUDbOem5BNkzkguRzln
CSqlG4iZFokCmAIvrhEWqWpEU33vfmAVOPYCH7rXk9%2BT4ge3wK0elHVb%2FfbGZcXJcHHQVdjJaUEPGdIABVehXjKP9
7E8o373MaP3%2BC%2F3hlml6hcgKDeVCMsJVrz3h93B%2BgbUxE%2FvOMx8EiApf94iNNvTs8FoZ9gM3Bvfv3U%2F0QyJ
I6bxAuu34fkcj3FTPakFSUJAOomjOtnkR1efOifFhPFQdP%2F4ik5JI%2BmvWUWPCNySYV%2FjKaQaGH1uD9RYrxTdDnq
IYZikmP9C%2B7US9mN6feo3oGhJv6g1Eke47R%2Bi46e6p%2BlQ%3D%3D
```

This unpacks to:

JWT Header:

```
{
  "typ": "JWT",
  "alg": "RS256"
  "kid": "cn=honeydo.flaminhoop.com; nva=2021-09-23 18:59:59 EST"
}
```

JWT Payload (direct-specified AB pattern):

```
{
  "iss": "urn:flaminhoop",
  "iat": 1554739158,
  "wmver": 1,
  "wmidfmt": "ab",
  "wmid": "ABBABAAABBABABBBAAABBABBAAA"
}
```

JWT Payload (encoded AB pattern in hex):

```
{
  "iss": "urn:flaminhoop",
  "iat": 1554739158,
  "wmver": 1,
  "wmidfmt": "hex",
  "wmid": "68EB8D8"
}
```

Signature:

```
a3lPLdeCWN3aa2Z3z1IBUDbOem5BNkzkguRzlnCSqlG4iZFokCmAIvrhEWqWpEU33vfmAVOPYCH7rXk9%2BT4ge3wK0el
HVb%2FfbGZcXJcHHQVdjJaUEPGdIABVehXjKP97E8o373MaP3%2BC%2F3hlml6hcgKDeVCMsJVrz3h93B%2BgbUxE%2Fv
OMx8EiApf94iNNvTs8FoZ9gM3Bvfv3U%2F0QyJI6bxAuu34fkcj3FTPakFSUJAOomjOtnkR1efOifFhPFQdP%2F4ik5JI
%2BmvWUWPCNySYV%2FjKaQaGH1uD9RYrxTdDnqIYZikmP9C%2B7US9mN6feo3oGhJv6g1Eke47R%2Bi46e6p%2BlQ%3D%
3D
```

In this case, the header confirms the format of the object, and the algorithm used to compute the signature. The payload carries the watermarking pattern (WMID) that the CDN should apply to the stream. The payload, sometimes referred to as “claim set” has a schema version that dictates how consumers of the information in the token should go about parsing it. This schema version field is “wmver” and is present in each and every WMT that will ever exist.

2.4.1.1 Payload Claims

This section goes into more detail on each of the claims that appear in the WMT.

Parameter	Claim Name	Allowable Values	Usage	Description
iat	Issued At	A number representing the numeric date of issue (e.g. UNIX timestamp)	Required	The "iat" (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the JWT.
iss	Issuer	A case sensitive string or URI	Required	The "iss" (issuer) claim identifies the principal that issued the JWT.
wmver	WMT Schema Version	1	Required	The “wmver” claim identifies the schema version of the WMT. This gives a hint to the parser as to the fields found in the token.
wmid	Watermarking ID (WMID)	A string that adheres to the format implied by the “wmidfmt” claim	Required	The “wmid” is the Watermarking ID that shall be used to indicate to the CDN Edge which variant should be delivered for the specific end user session.

wmidalg	WMID Decryption Algorithm	aes-128-cbc, aes-256-cbc	Optional	The “wmidalg” claim identifies the cipher algorithm to use when decryption the pattern.
wmidfmt	WMID Format	ab, hex	Optional	The “wmidfmt” claim identifies the encoding format of the WMID pattern. ab: specifies the pattern is a sequence of “A” and “B” characters. E.g. `"wmid": "ABBABAAABBBABABBBAAABBABBAAA"` The pattern is interpreted by the CDN from left to right, similar to the way an index into an array would work. E.g. an index of 3 would return “A”. hex: specifies the pattern is a hex string when decoded into binary implies 0=A and 1=B. E.g. `"wmid": "68EB8D8"` which translates to `"ABBABAAABBBABABBBAAABBABBAAA"`. The hex string should be treated as a sequence of characters that expand to their AB equivalents, e.g. “6” would be ABBA (0110), where an index of 3 into “0x6” would be “A”. Default: “ab”
wmidivlen	WMID IV Length	1..N	Optional	The “wmidivlen” claim identifies the length of the Initialization Vector (IV) suitable for the specified decryption algorithm (wmidalg).
wmidivhex	WMID IV Hex String	Hexadecimal String	*	The “wmidivhex” claim identifies the IV to use with the specifieid decryption algorithm. *This claim is required if the wmidalg cipher requires an IV, and the target system must receive a Hex String encoded value.
wmidivb64	WMID IV Base64	Base64 String	*	The “wmidivb64” claim identifies the IV to use with the specified decryption algorithm. *This claim is required if the wmidalg cipher requires an IV, and the target system must receive a Base64 encoded value.
wmidoff	WMID Pattern Offset	128, 256	Optional	The “wmidoff” claim identifies an offset that should be applied to the function that calculates the location in the pattern to use

				for the current MSN. The offset allows for a prohibitively large pattern to be defined, but only a portion of it sent in the WMT at any given point in time. The legal values are intentionally large to allow for WMT caching to be effective.
wmidpid	WMID Password ID	String	*	The “wmidpid” claim identifies the password to use when generating the pattern decryption key. *This claim is required if the wmidalg cipher requires a decryption key
wmidpalg	WMID Password Algorithm	sha256	*	The “wmidpalg” claim identifies the hashing function to use when generating a decryption key from a password. *This claim is required if the wmidalg cipher requires a decryption key

2.4.2 Token Security

In order to prevent tampering of the watermarking function, the pattern must be delivered to the CDN in a secure fashion. To that end, a signing algorithm must be used on the token. In the previous example, asymmetric PKI was used using the private key of the web server.

2.4.3 Pattern Encryption

An optional layer of security would be to encrypt the pattern (WMID) such that savvy end users cannot see it. In the proposed Portal behavior, the Pattern Encryption section deals with the necessary parameters to decrypt a pattern encrypted with AES-128-CBC/AES-256-CBC. For example:

JWT Payload:
```
{
  "iss": "urn:flaminhoop",
  "iat": 1561761145,
  "wmver": 1,
  "wmidalg": "aes-128-cbc",
  "wmidivlen": 16,
  "wmidivhex": "a45890072f06aebaa4786fb540ab707a",
  "wmidctb64": "5hOdS05QcLFVSyjlZnF9mDGR1ipqw949MqYfanFIyMI=",
  "wmidpid": "decryptpw_2017-06-28",
  "wmidpalg": "sha256"
}
```

This specifies that the incoming pattern has been encrypted with AES-128-CBC using a key derived from a SHA-256 hash of a password identified by the label “decryptpw_2017-06-28”. In this scenario, both Akamai and the Watermarking Vendor would configure the password identified by the label in their respective systems. Then, the Watermarking Vendor would scramble the pattern in the following way:

```
KEY = HASH( password )
   Password identified by wmidpid
   Hash algorithm identified by wmidpalg
IV = generate random IV of length suitable for algorithm identified by wmidalg
CIPHERTEXT = AES-128/256-CBC( KEY, IV, pattern )
```

2.4.4 Token Location

The Watermarking Token (WMT) can be sent to Akamai in three different locations:

1. Authorization Header
2. Query Parameter
3. Virtual Directory

2.4.4.1 Authorization Header

The Authorization header is the recommended method to send the WMT from client to Edge. The format is straightforward:

    Authorization: <wmt>

The client application has the responsibility of inserting the Authorization header to (at least) all content requests made of the CDN.

2.4.4.2 Query Parameter

The query section of the URL can also be used to send the WMT to Edge. The query parameter name “wmt” is reserved for this purpose. For example:

    https://ott-provider.akamaized.net/title1/1080/segment_1.ts?wmt=WMT

The client application has the responsibility of inserting the “wmt” query parameter to (at least) all content requests made of the CDN.

2.4.4.3 Virtual Directory

A Virtual Directory (VD) can be used to send the WMT from client to Edge. The VD approach, when combined with HLS or DASH and relative paths, has the unique benefit of allowing a URL provider (e.g. CMS) to give the client a URL with a WMT embedded in it, and for that WMT to included in all Edge requests from there on in. An example of this benefit can be seen in the following section: 2.3.1 Playback URL Generation: Token as a Virtual Directory.

3 References

1. Streaming Video Alliance: Combating Piracy with Forensic Watermarking
2. RFC 7515: JSON Web Signature (JWS)
3. RFC 7516: JSON Web Encryption (JWE)
4. RFC 7517: JSON Web Key (JWK)
5. RFC 7518: JSON Web Algorithm (JWA)
6. RFC 7519: JSON Web Token (JWT)
7. RFC 7520: JOSE Examples
8. Base64 Decoder
9. JWS/JWE for Not so Dummies

4 Version History

1.8 - 2019-12-12

- Added some DASH requirements

1.6 - 2019-10-07

- Added important sequence diagrams to help illustrate both token and segment retrieval workflows
- Documented supported token locations
- Added more description to the Architectural Overview

1.5 - 2019-09-19

- Updated origin format to use the "movie/b.segment.ts" method as opposed to a virtual directory in front of the segment, e.g. "movie/B/segment1.ts".
- Adding some clarification on how the pattern (WMID) is to be indexed into

1.4 - 2019-08-21

- Fixed spelling error in subtitle

1.3 - 2019-07-11

- Updated the JWT header and payload claims to clarify how cipher algorithms (signature verification, optional pattern decryption) are specified
- Added discussion on how pattern encryption can be performed

- Reverted WMT schema (wmver) back to 1 and introduced a claim to define what the pattern format is, either “ab” or “hex”.
- Added a table describing WMT claims