DASH-IF Implementation Guidelines: Token-based Access Control for DASH (TAC)

July 20, 2016

DASH Industry Forum

Draft Version 1.0 (Community Review)

Scope

The scope of this document is to define a token-based access control mechanism and enabling the signaling of authentication and authorization (AA) protocols for DASH-based video streaming. An access token is a proof that a DASH client or user of the clients have been successfully authenticated and authorized, respectively, in some pre-determined AA Systems to access a particular DASH resource, e.g. DASH segments or MPDs.

This document remains agnostic on what AA Systems and protocols are actually used for DASH clients to obtain AA Tokens but it does define the access token format for accessing DASH resources, hence ensuring interoperability between content providers and content delivery networks.

Disclaimer

This document is not yet final. It is provided for public review until October 31st, 2016. If you have comments on the document, please submit comments at the following URL:

- at the github repository https://github.com/Dash-Industry-Forum/TAC/issues
- at the public repository https://gitreports.com/issue/Dash-Industry-Forum/TAC

Please add a detailed description of the problem and the comment.

Based on the received comments a final document will be published latest December 31st, 2016, integrated in a new version of this document.

This is a document made available by DASH-IF. The technology embodied in this document may involve the use of intellectual property rights, including patents and patent applications owned or controlled by any of the authors or developers of this document. No patent license, either implied or express, is granted to you by this document. DASH-IF has made no search or investigation for such rights and DASH-IF disclaims any duty to do so. The rights and obligations which apply to DASH-IF documents, as such rights and obligations are set forth and defined in the DASH-IF Bylaws and IPR Policy including, but not limited to, patent and other intellectual property license rights and obligations. A copy of the DASH-IF Bylaws and IPR Policy can be obtained at http://dashif.org/.

The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS, and the authors and developers of this material and DASH-IF hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of workmanlike effort, and of lack of negligence.

In addition, this document may include references to documents and/or technologies controlled by third parties. Those third party documents and technologies may be subject to third party rules and licensing terms. No intellectual property license, either implied or express, to any third party material is granted to you by this document or DASH-IF. DASH-IF makes no any warranty whatsoever for such third party material.

Contents

1 Introduction

1.1 General

Common DASH use cases may require authentication of the end user or their player/device, followed by authorization to access the content described in an MPD. The authentication and authorization operations are commonly performed by Authentication and Authorization (AA) systems. Because authorization depends on authentication, the two functions are usually performed sequentially starting with authentication and then authorization.

Example scenarios where authorization is useful are:

- Streaming is restricted to a geographic region where the service provider has distribution rights.
- Streaming is restricted to trusted DASH clients that present ads in the video, MPD, or signaled for insertion; and accurately report playback.
- Streaming is restricted to end users who are subscribers or have purchased rental/ownership for streaming/download of SD/HD/UHD quality, for a particular date range or number of views, for particular devices and protection systems, for a maximum number of devices or simultaneous streams, etc.
- Streaming may be enabled using federated identity systems such as TV Everywhere, UltraViolet, OpenID, and various SSO systems (Single Sign On), and federated rights systems such as DECE, KeyChest, TV Everywhere, etc.

This document defines the signaling, the exchange mechanism of access tokens and the access token format for the purpose of granting the DASH clients access to DASH resources, e.g. DASH segments or MPDs. Typically, authentication and authorization information takes the form of AA tokens as proofs that the clients or the users of the clients have been authenticated and authorized according to some pre-determined AA systems (or schemes). These token formats are out-of-scope and any existing AA schemes may be used. Essentially, the goal of this document is to enable interoperability at the resource access level between DASH clients and HTTP servers delivering DASH content while allowing various AA schemes to be used for obtaining the access control token.

The signaling and the exchange mechanisms defined in this document leverages on the AA signaling and information exchanging mechanisms defined in ISO/IEC 23009-1:2014/ Amd3:2016.

1.2 References

1.2.1 Normative References

[DASH] ISO/IEC 23009-2:2014 Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats.

[DASH-AMD3] ISO/IEC 23009-1:2014/Amd 3:2016, FDAM, February 2016.

[URISigning] URI Signing for CDN Interconnection (CDNI), K. Leung, F. Le Faucheur, R. van Brandenburg, B. Downey, M. Fisher, April 5, 2016, source: https://tools.ietf.org/html/draft-ietf-cdni-uri-signing-07

[URISigning-HAS] URI Signing for HTTP Adaptive Streaming (HAS), R. van Brandenburg, June 6, 2016, source: https://tools.ietf.org/html/draft-brandenburg-cdni-uri-signing-for-has-03

[Base64] The Base16, Base32, and Base64 Data Encodings, S. Josefsson, RFC4648, October 2006

1.2.2 Informative References

[DASH-264] DASH-IF Guidelines for Implementation: DASH264/AVC Interoperability Points, May 2015.

[OMAP] Online Multimedia Authorization Protocol. Open Authentication Technology Committee. Version 1.0, August 22, 2012.

[OAuth] The OAuth 2.0 Authorization Framework, D. Hardt, Ed., October 2012, RFC 6749, source: https://tools.ietf.org/html/rfc6749

[CPA] ETSI TS 103 407: Cross Platform Authentication for limited input hybrid consumer equipment, source: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=47970

1.3 Terms & Definitions

AA Token

authentication token or authorization token

Access Token

token granting, if valid, the access of a resource identified by an URL

Authentication

process of determining whether a user or client is, in fact, who or what it is declared to be, which may rely on some type(s) of user and/or client identification and credentials

Authentication Token

token as a proof of being authenticated

Authorization

process of determining whether which a user or client is authorized or has permissions to access content, which may rely on authenticating the user or client

Authorization Token

token as a proof of being authorized

Content

One or more audio-visual elementary streams and the associated MDP if in DASH format

Token

form of credentials or proof, used for some purpose, encoded digitally as an opaque string.

2 Use Cases for AA System and their requirements

2.1 Introduction

Authentication and Authorization information needs to be exchanged between DASH system entities, typically servers and clients, in order to allow DASH clients to access protected content. Typically, the AA information takes the form of AA tokens. Upon reception of valid AA Tokens, the client obtains an Access Token. Consequently, when requesting segments or MPDs, the client provides the Access Token along with the requests. When the Access Token is valid, the server should grant the access to the content and deliver the resource to the client. Note that the AA Systems and workflows for generating these Access Tokens may vary and depend on a number of factors, such as available client identification and credentials, server requirements and security measures as well as the type and quality of content that can be On Demand or Live, and SD, HD or UHD.

This section presents different use cases where such Access Token generations and exchanges are required.

2.2 Token concept and definition (informative)

An Access Token is a merely a proof of one or more past actions granting access to a resource. Figure 1 shows the concept of protecting a resource by a token.

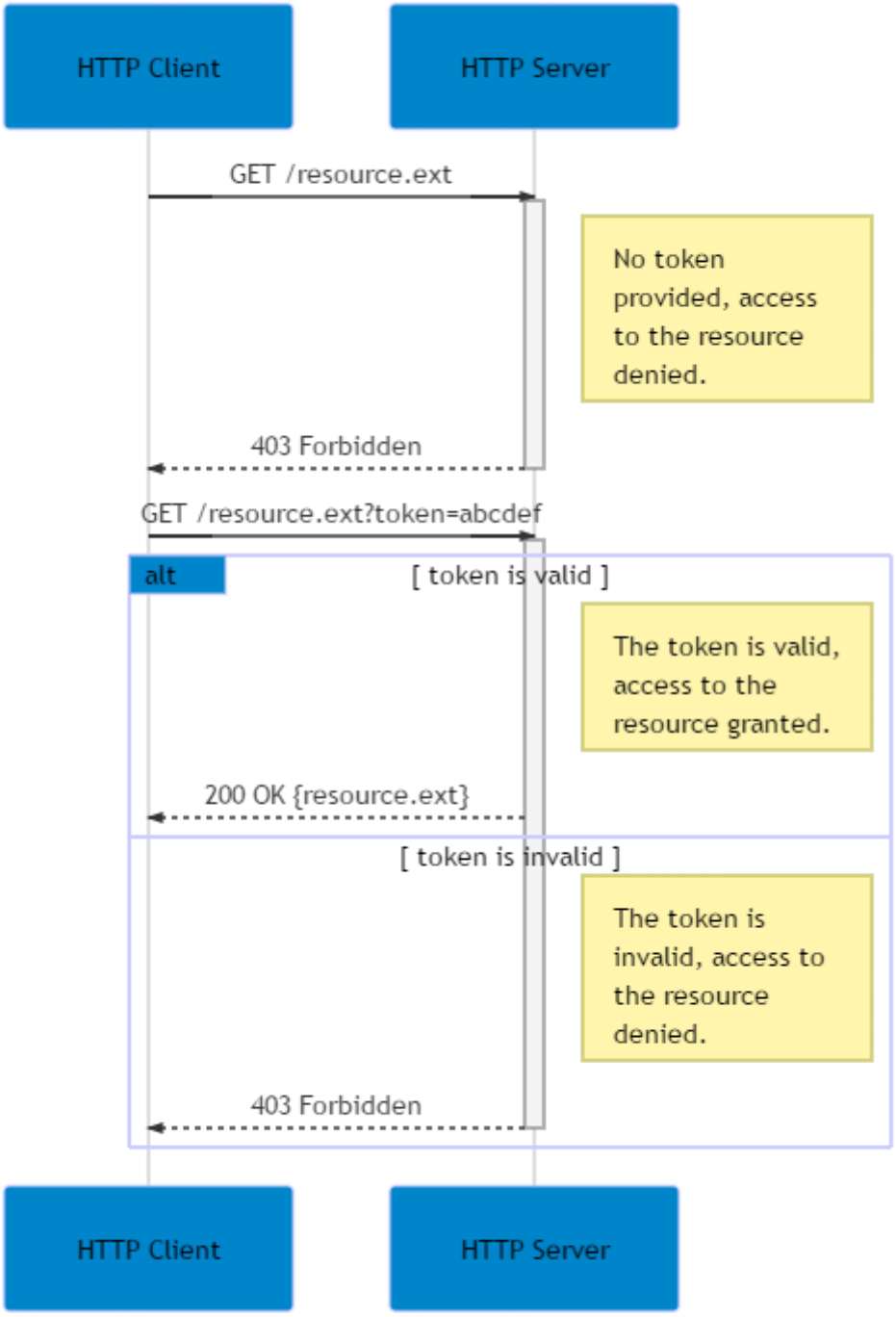

Figure 1 - Token-protected resource retrieval

A client is sending an HTTP request to an HTTP server. If the HTTP request does not contain a valid token or any token at all, the HTTP server does not serve the request. On the contrary, if a valid token is provided in the HTTP request, the HTTP server delivers the requested resource. In the context of DASH, the resource can be for instance MPDs or media segments.

A token can be characterized by two main aspects:

- The required action to obtain the token
- The rules that determine the validity of the token

When the action to obtain a token involves authenticating a client, this token is usually called Authentication Token. Similarly, when the action to obtain a token involves authorizing a client, this token is usually called Authorization Token. Note also that these two steps are commonly chained, i.e. a client needs to provide an Authentication Token before requesting an Authorization Token.

Regarding the validity rules of a token, they typically involve a validity time, a resource identifier, e.g. an URL, an IP address of the allowed clients, etc. Nevertheless, the DASH client is agnostic as to what the token holds as information and merely sees it as an opaque string. Only the entity issuing the token and the one validating it must understand the token format.

2.3 Overview of AA System Architecture (informative)

This informative section gives an overview of the functions involved in AA System. The DASH client and/or the application around is able to request and retrieve a token that grants him access to the requested resource such as segments, MPD, etc.

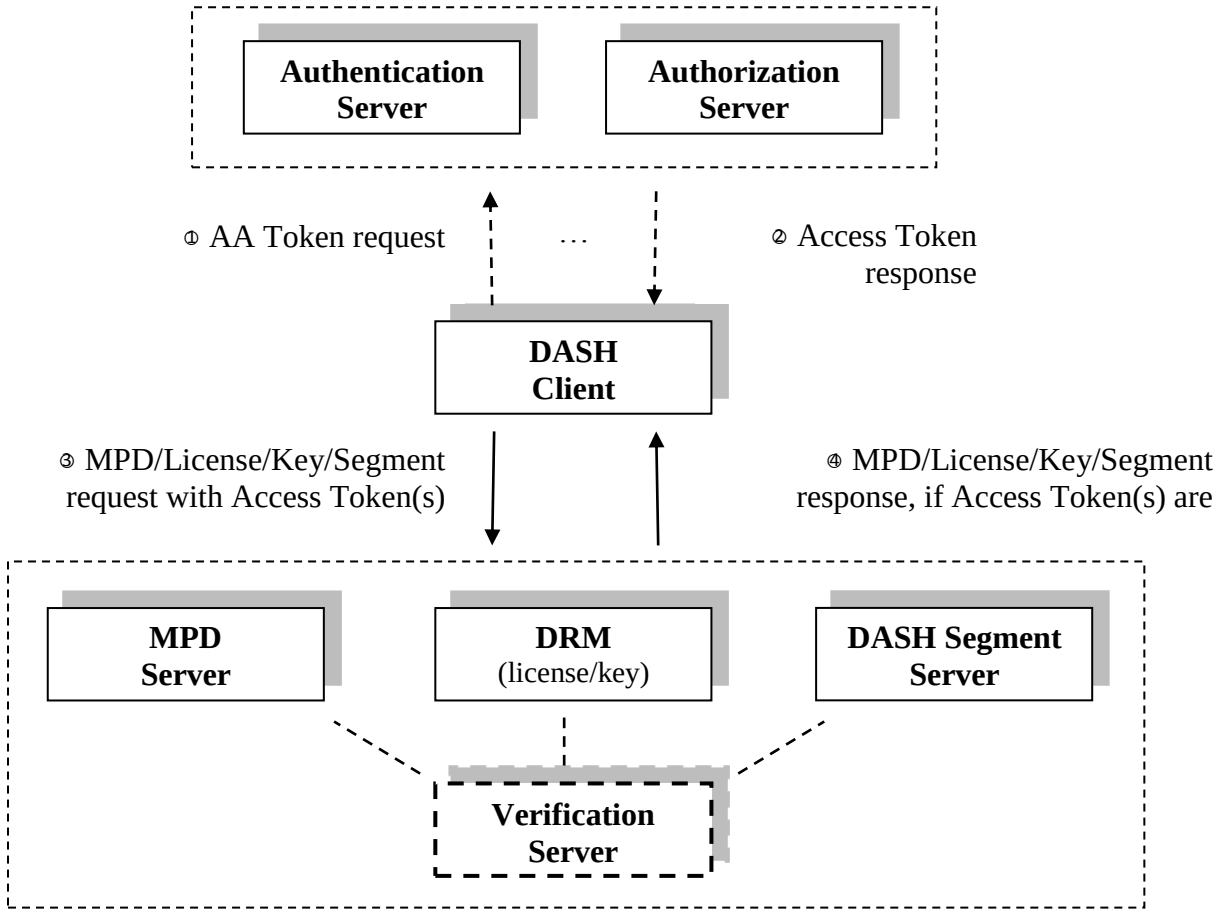

Figure 2:AA System Architecture

Figure 2 shows logical entities that may request, issue, provide and verify token-based AA information for the purpose of granting access to requested MPD, DRM licenses, crypto keys and content segments. A physical entity may combine multiple logical roles, and a logical role can be played by more than one physical entities (e.g., accessing segments of different types and qualities may be authorized by different Authorization Servers and verified by different Verification Servers). The point of origin for information (e.g., credentials and protocols used for obtaining tokens) and information contained within tokens can differ; so various information

flows in requesting and generating tokens are possible. Because of this, the document focuses on signaling and exchange mechanisms to facilitate Access Token based requests and delivery for MPDs, licenses, keys and content segments (steps 3 and 4 in Figure 2).

The roles of the entities in Figure 1 are:

Authentication Server – A service that issues authentication tokens.

Authorization Server– A service that issues authorization tokens.

DASH Client – A client that makes requests for DASH content segments.

MPD Server – A service that delivers DASH MPDs

DRM (License/Key) Server – A service that provides DRM information and data such as DRM licenses and crypto keys

DASH Segment Server – A service that provides DASH content segments.

Verification Server – A function that verifies Access Tokens.

2.4 Use Cases for DASH resource access control

Use cases can be characterized by the following aspects of Access Tokens:

- Issuer: servers, or authorities
- Receiver: clients, users, or CDN nodes
- Duration: short-term, long-term, or periodically refreshed
- Scope: MPD, periods, content types, adaptation sets, representations, or individual segments
- Scheme: single or multiple AA schemes
- Signaling: MPD, in-band or out-of-band

2.4.1 Use case 1: Mandatory pre-roll downloading

A service provider offers MPDs that contain two Periods, i.e. one for a pre-roll advertisement and one for the main video. The service provider wishes that the DASH clients must first download the pre roll advertisement before being able to retrieve the main content. This way, the end users must return to the ad server before viewing additional content. To this end, an Access Token protects the delivery of the main content. Only the segment requests providing a valid Access Token are granted. In addition, the service provider configured the HTTP server delivering the advertisement to insert the segment Access Token for the main content in the HTTP response of the advertisement segment.

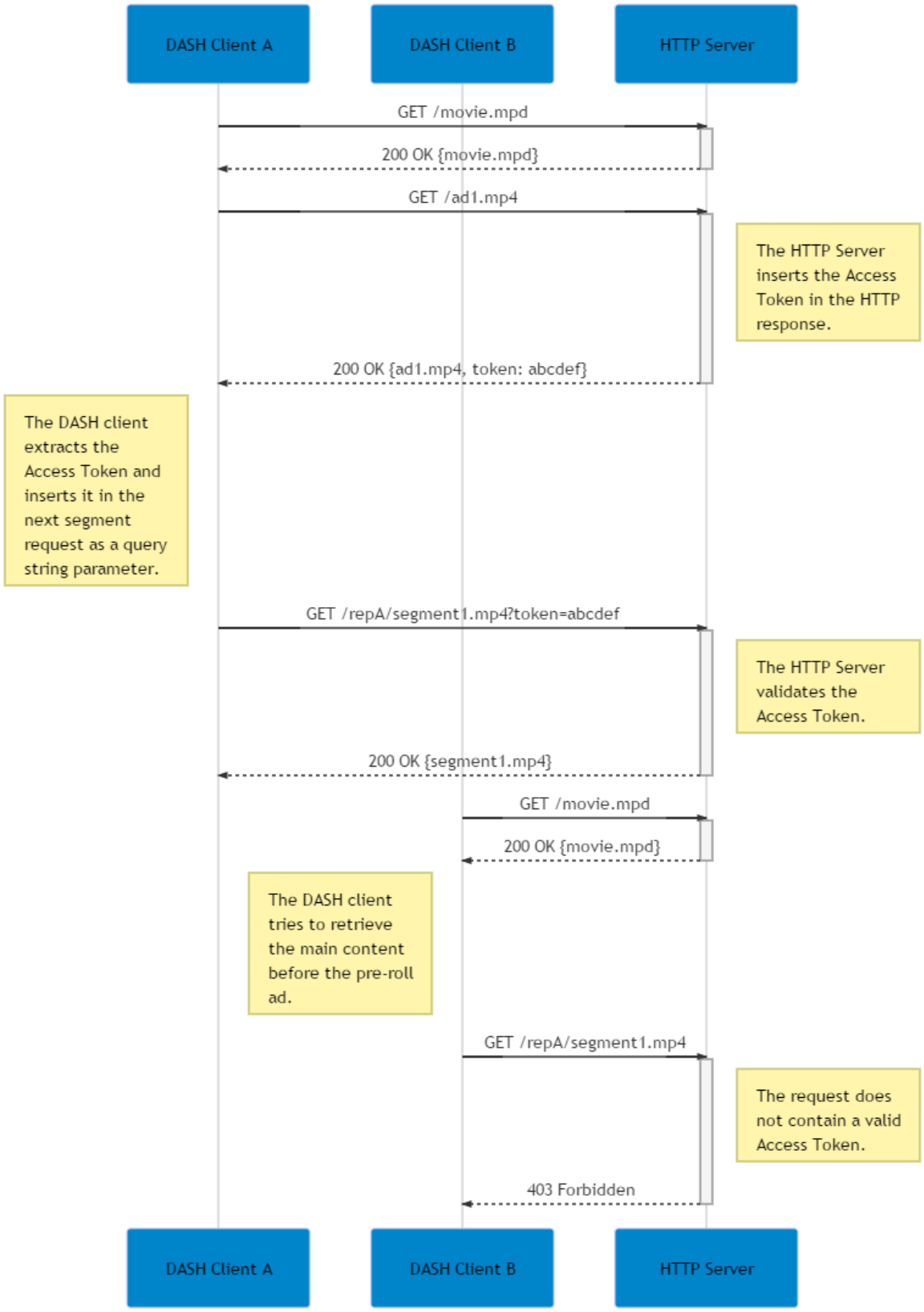

Figure 3 - Ad pre roll download sequence

2.4.2 Use case 2: Ad free premium service

In this use case, the same service provider as in use case 1 wants to offer the possibility to pay a small monthly fee to be able to skip the content. The end user must authenticate himself/herself in the portal and subscribe to the ad free feature. Upon MPD download, the

MPD server provides the MPD along with the segment access token generated by the Authorization Server shortly before. Note that, in this case, the MPD only contains the main video without the advertisement period as in the previous use case.

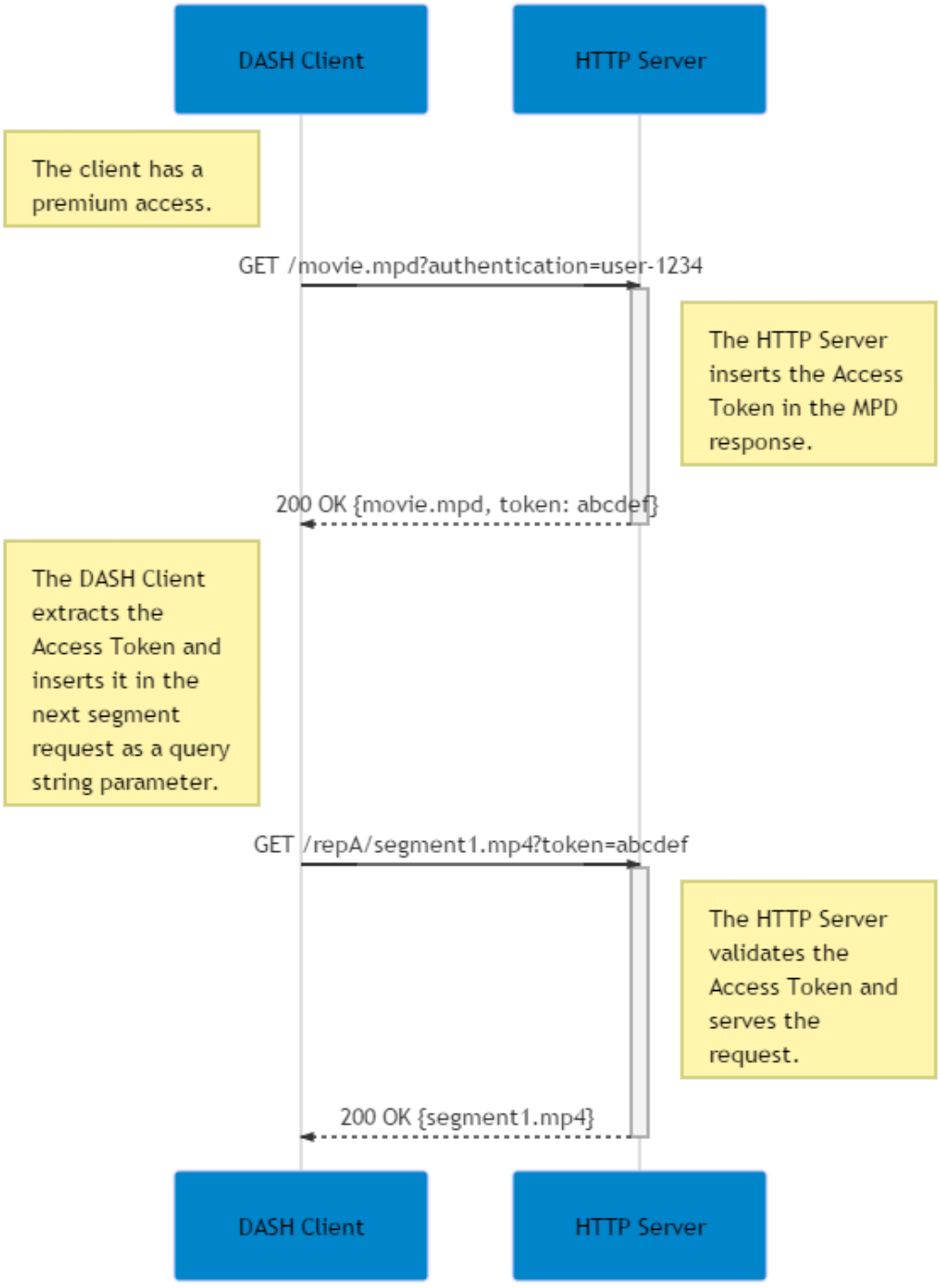

Figure 4 - Ad free viewing for premium client sequence

2.4.3 Use case 3: Service provider using third party CDNs

A service provider wants to manage its movie portal but outsources the delivery of the MPD and the segments to a third party CDN. In this case, the CDN has no business logic to decide whether a DASH client requesting an MPD or a segment is authorized to retrieve the content. However, a CDN may validate whether a token is valid using the signed information it contains, e.g. the expiration time, client IP address, etc. The end user first purchases a movie via the

portal of the service provider. Upon success, the portal gives the MPD URL as well as a short-term Access Token to the application. For security reason, this Access Token is a short-term token, i.e. it expires couples of seconds after its generation. This way, it mitigates the risk of unauthorized clients using the token after its generation. Consequently, the application sends a MPD request to the CDN along with the Access Token. The CDN verifies the validity of the Access Token and delivers a new Access Token inside the MPD response for the DASH client to access the segments. This new Access Token has a longer validity period compared to the one for the MPD and thus is called a long-term token. Every time the DASH client requests a segment it uses the same long-term token.

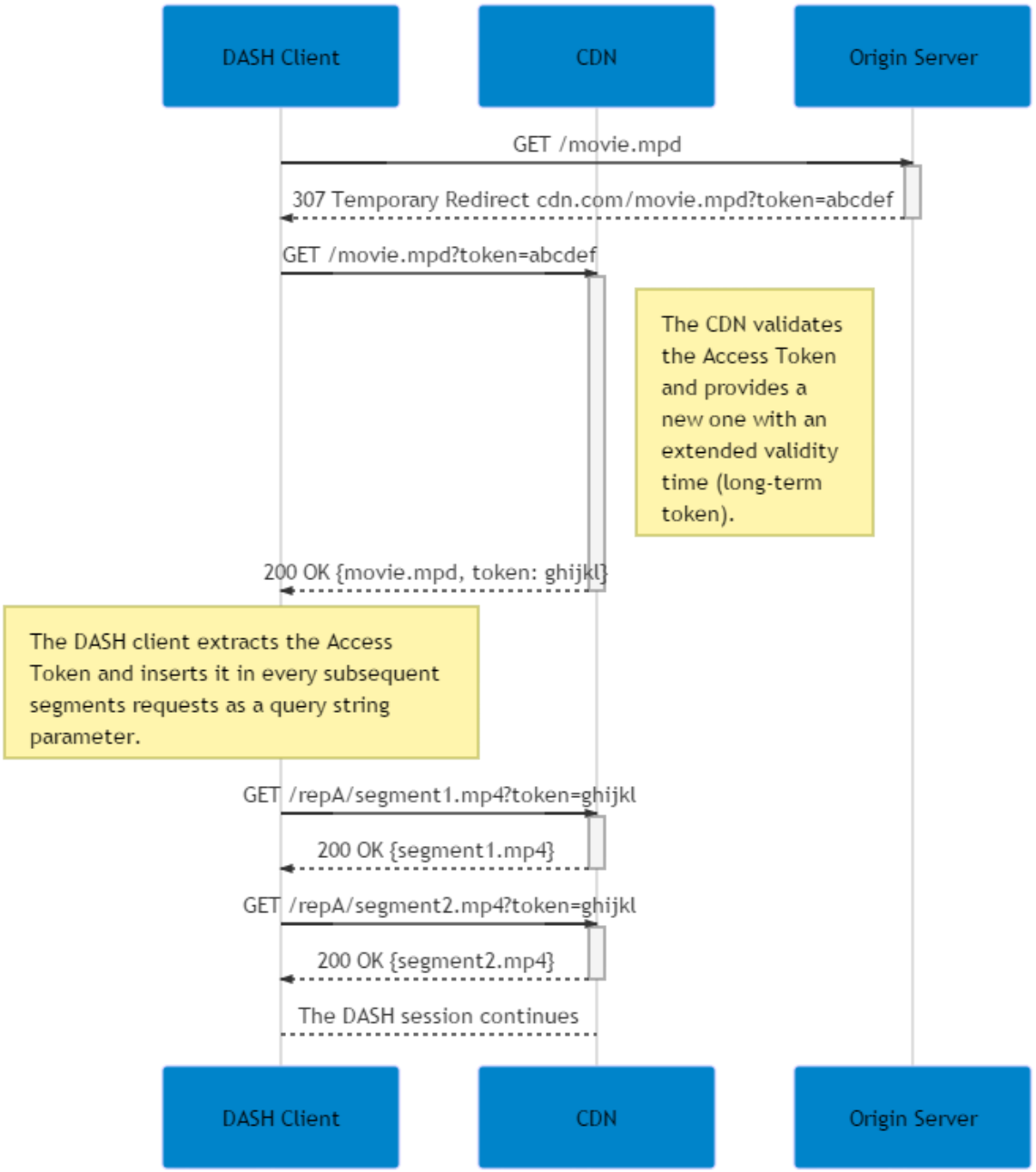

Figure 5 - Segment access token with Origin Server and CDN integration sequence

2.4.4 Use case 4: DRM license retrieval protection

A DRM system provides an API to download license keys for decrypting DASH content. Only authorized clients get an Access Token to retrieve the license key. To this end, the HTTP server

hosting license key is configured to require the presence of a valid Access Token in the request for a license key. The same workflow as in 2.2 can be applied to the protection of the license key retrieval.

3 Access Token format

This section specifies the format of the Access Token.

The URI Signing [URISigning] defines a token format for protecting the access to general URL resources. The extension of the URI Signing for HTTP Adaptive Streaming [URISigning-HAS] extends the URI Signing token format with specific parameters pertaining to segmented content protection, especially for adaptive streaming use cases.

The Access Token shall be formatted as the Signed Token defined in [URISigning-HAS]. In addition, the Signed Token Transport (STT) parameter of Access Token shall be set the a value of 2.

As a result, the Authorization Server shall use this format when generating tokens protecting DASH resources. Consequently, the Verification Server shall use this format when validating the received tokens.

An overview of the Signed Token is provided in Annex A for information.

4 Access Token transport

This section specifies the way to transport the Access Token.

4.1 Access Token transport format

The Access Token shall be base64 encoded according to Base-64 Data Encoding [Base64] when being transported.

The Authorization Server shall then base64 encode the Access Token after its generation. Consequently, the Verification Server shall base64 decode the received token to recover the Access Token.

4.2 Access Token transport mechanism

The Access Token shall be transported in a HTTP header field when communicated in a HTTP 2xx Successful message.

The Access Token shall be transported in a query string parameter when communicated in a HTTP 3xx Redirection message or in a HTTP request.

The following ABNF syntax for the header field shall be used:

```
DASH-header-field = "DASH-IF-IETF-Token" ":" token

token = access-token
```

The field `access-token` is a string containing the base64-encoded Access Token.

The query string parameter name shall be "`dash-if-ietf-token`" whose value is the base64-encoded Access Token.

NOTE – Although HTTP header names are case insensitive, the typography used above helps the distinction between the HTTP header name and the query string parameter name.

NOTE – Future token format may be added a later point in time by defining new combinations of HTTP custom header name and query string parameter names.

4.3 Registration of transport mechanism

The [URISigning-HAS] specification provides the ability to extend the specification with additional transport mechanism via the parameter Signed Token Transport (STT) (see A.2).

The value "2" is registered as the "DASH-IF Token Transport"

[Editor's note] The CDNI working group has not yet created the registry. The actual value may change upon registration.

4.4 Note on cookie-based transport

Using cookies to communicate tokens in general can be problematic for the following reasons:

- Some embedded devices do not use the same User Agent to get the MPD and the segment, resulting in cookie not found.

- When content providers use multiple CDNs or deliver MPDs from a domain other than the segments, cookies are not delivered to the CDN since cross-domains cookie is not supported.
- Cookie support across browsers varies to such a degree that even across versions of the same browser may differ significantly

Therefore, the transport of the Access Tokens specified in this document does not rely on cookies but relies on HTTP header extensions and on query string parameters, see section 5.

5 Access Token exchange protocol

This section specifies the successive steps for the exchange of the Access Token.

The specification enables three types of mechanisms for a DASH Client to retrieve the initial Access Token. The MPD author may choose one of these three types.

The first mechanism instantiates the initial Access Token from regular DASH operations, e.g. MPD request, segments requests, Xlink resolution etc. In practice, the HTTP server delivers the requested resource along with the Access Token. In addition, the MPD author signals in the MPD how to extract the Access Token and from which HTTP responses.

The second mechanism instantiates the initial Access Token from an appropriate XML element in the MPD in. In some situations, the MPD may not be delivered via HTTP or the MPD author wishes to embed the initial Access Token within the MPD itself. In this case, this mechanism is suitable.

NOTE – In both first and second mechanisms, the DASH Client is agnostic as to what the nature of the Access Token is and merely sees it as an opaque string that needs to passed along with future HTTP requests.

The last mechanism instantiates the Access Token via an external protocol. In this case, the MPD signals the protocol to be used by the application to retrieve the initial Access Token. When the application recognizes one of the signaled protocols, it executes the corresponding protocol as specified by the scheme. Since out-of-scope of this specification, it is expected that the application implements the different steps of the protocol that leads to the retrieval of token

5.1 HTTP-based instantiation

For this type of Access Token instantiation, the Access Token is delivered to the DASH Client via regular DASH operations.

For instance, the Access Token may be delivered in a HTTP header of a MPD response, a segment response, a xlink response, etc... Consequently, the MPD author needs to instruct the DASH Client to extract the Access Token. To this end, the **`ExtUrlQueryInfo`** shall be used and should be configured according to the mechanism expected by the MPD author.

The example MPD below signals the presence of the Access Token in the HTTP responses for MPD requests (`@headerParamSource` attribute) inside the HTTP custom header called "DASH-IF-IETF-Token" and instructs the DASH Client to insert this Access Token into segment and MPD requests within the "dash-if-ietf-token" query string parameter.

```
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1" maxWidth="1280"
maxHeight="720" maxFrameRate="25" par="16:9">
  <EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2016:querystring"
xmlns:up="urn:mpeg:dash:schema:urlparam:2016">
    <up:ExtUrlQueryInfo
        headerParamSource="mpd"
        includeInRequests="segment mpd"
        queryTemplate="dash-if-ietf-token=$header:DASH-IF-IETF-Token$"/>
  </EssentialProperty>
  <SegmentTemplate duration="2" startNumber="1" media="seg$Number$.mp4">
  </SegmentTemplate>
    <Representation id="v0" codecs="avc3.4d401f" width="1280" height="720" frameRate="25"
sar="1:1" bandwidth="3000000"/>
</AdaptationSet>
```

In the example above, the DASH Client is expected to perform the following steps according to [DASH-AMD3] when sending HTTP requests for MPD and segments:

Step	Action	URL construction
1	Determine the URL of the resource	http://cdn.com/movie/seg1.mp4
2	Insert the query string part in the requested URL according to the `@queryTemplate` attribute	http://cdn.com/movie/seg1.mp?dash-if-ietf-token=$header:DASH-IF-IETF-Token$

3	Determine the most recent value of the HTTP header 'Access-Token' received in a MPD response.	`HTTP/1.1 200`
		`DASH-IF-IETF-Token: rtziwO2HwPfWw~yYD`
		
		`<MPD>`
		…
		`</MPD>`

4	Substitute the expression in the query template	http://cdn.com/movie/seg1.mp?dash-if-ietf-token=rtziwO2HwPfWw~yYD

NOTE –The body of the HTTP response can be cached by the CDN edge in this case since the MPD is not client specific. Only the HTTP headers for the Access Token is computed on-the-fly which is customary for many of the HTTP headers in HTTP responses.

5.2 MPD-based Access Token instantiation

The Access Token may be inserted in the MPD. It is encouraged to secure the delivery of the MPD in that case via a secure transport protocol, e.g. HTTP over TLS, to prevent illegal clients from intercepting the Access Token.

The `@queryString` attribute of either the **`UrlQueryInfo`** or **`ExtUrlQueryInfo`** descriptors shall be used to carry the Access Tokens.

Below is an example using the **`ExtUrlQueryInfo`** descriptor:

```
<EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2016:querystring"
xmlns:up="urn:mpeg:dash:schema:urlparam:2016">
    <up:ExtUrlQueryInfo
        includeInRequests="mpd segment"
        queryString="token=nitfHRCrtziwO2HwPfWw~yYD"
        queryTemplate="dash-if-ietf-token=$query:token$"/>
  </EssentialProperty>
</EssentialProperty>
```

NOTE – The query string parameter in the `@queryString` attribute is not normatively defined. The MPD author may choose the query string parameter name it wishes.

When using the **`UrlQueryInfo`** descriptor, the echoing mechanism described in "5.4 Access Token " is not possible.

This **`EssentialProperty`** should be located in the appropriate level in the MPD, for instance in a Representation, an AdaptationSet, etc... In the example above, the DASH Client is expected to perform the following steps according to [DASH-AMD3] when sending HTTP requests for MPD and segments:

Step	Action	URL construction
1	Determine the URL of the resource	http://cdn.com/movie/seg1.mp4
2	Insert the query string part in the requested URL according to the `@queryTemplate` attribute	http://cdn.com/movie/seg1.mp?dash-if-ietf-token=$query:token$
3	Substitute the expression in the query template, i.e. extracting the value of the parameter access-token in the `@queryString` attribute	http://cdn.com/movie/seg1.mp?dash-if-ietf-token=nitfHRCrtziwO2HwPfWw~yYD
4	Send the HTTP request with the final URL	GET http://cdn.com/movie/seg1.mp?dash-if-ietf-token=nitfHRCrtziwO2HwPfWw~yYD$

5.3 External Access Token instantiation

The external instantiation of Access Tokens comprises the following additional aspect:

- Use of the Authentication and Authorization descriptors in [DASH-AMD3],
  - to signal any AA schemes and possibly parameters needed to obtain Access Tokens, and
  - to carry any available Access Tokens or to signal any URLs where Access Tokens can be retrieved.

The table below constitutes the only normative aspect introduced by this paragraph "5.3 External Access Token instantiation"

Table 1 - Parameter identifier for substitution in query string template

$<Identifier>$	Substitution parameter
`$AASchemeIdUri$`	The identifier shall be substituted by the scheme identifier of the **`EssentialProperty`** whose attribute `@id` value was "`mpeg:dash:content-authorization:2014`" that was used to obtain the Access Token.
`$AccessToken$`	The identifier shall be substituted by the Access Token obtained via an AA System.

The Client Authentication and Content Authorization descriptors in [DASH-AMD3] provide a mechanism to signal identification information of AA Systems. The following examples illustrate how client authentication and content access authorization information is signaled in the MPD.

```
<EssentialProperty schemeIdUri="urn:org:example:plan-a"
id="mpeg:dash:client-authentication:2014"
value="http://domain.com/authenticationServerA/protocolA?=ServiceSpecificInfoA"/>

<EssentialProperty schemeIdUri="urn:org:example:plan-b"
  id="mpeg:dash:client-authentication:2014"

value="http://domain.com/authenticationServerA/protocolB?=ServiceSpecificInfoB"/>
```

```
<EssentialProperty schemeIdUri="urn:org:example:plan-c"
   id="mpeg:dash:content-authorization:2014"

value="http://domain.com/authorizationServerC/protocolC?=ContentSpecificInfoC"/>

<EssentialProperty schemeIdUri="urn:org:example:plan-d"
   id="mpeg:dash:content-authorization:2014"

value="http://domain.com/authorizationServerD/protocolD?=ContentSpecificInfoD"/>
```

The application on top of the DASH Client is expected to perform the following actions.

Step	Action
1	Find the supported Client Authentication descriptors if any
2	Execute one of the supported Client Authentication protocol whose endpoint is signaled in the `@value` attribute
3	Retrieve the Authentication Token as proof of a successful client authentication
4	Find the supported Content Authorization descriptors if any
5	Execute one of the supported Content Authorization protocol whose endpoint is signaled in the `@value` attribute. It may require to provide the Authentication Token from step 3, if present, depending on the actual protocol.
6	Retrieve the Access Token

The **`UrlQueryInfo`** provides the DASH Client with the query string to append in the segment URL:

```
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1" maxWidth="1280"
maxHeight="720" maxFrameRate="25" par="16:9">

  <EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2014"
xmlns:up="urn:mpeg:dash:schema:urlparam:2014">
    <up:UrlQueryInfo queryTemplate="system=$AASchemeIdUri$&token=$AccessToken$"/>
  </EssentialProperty>

  <SegmentTemplate duration="2" startNumber="1" media="seg$Number$.mp4"/>

  <Representation id="v1" codecs="avc3.4d401f" width="640" height="360" frameRate="25"
sar="1:1" bandwidth="1500000"/>

</AdaptationSet>
```

NOTE – Since this instantiation use external protocols and token formats, there is no normative HTTP header name nor query string names to use. This is up to the MPD author to choose them.

The DASH Client is then expected to perform the following steps:

Step	Action	URL construction
1	Determine the URL of the segment	http://cdn.com/movie/seg1.mp4
2	Insert the query string part in the requested URL	http://cdn.com/movie/seg1.mp?system=$AASchemeIdUri$&token=$AccessToken$

	according to the `@queryTemplate` attribute	
3	Substitute `$AASchemeIdUri$` and `$AccessToken$` with the AA scheme used and the Access Token retrieved from it.	http://cdn.com/movie/seg1.mp?system=urn:org:example:plan-c&token=PfWw~yYD
4	Send the HTTP request with the final URL	GET http://cdn.com/movie/seg1.mp?system=urn:org:example:plan-c&token=PfWw~yYD

.

5.4 Access Token refresh

For enabling the Access Token refresh, the descriptor **`ExtUrlQueryInfo`** shall be used in combination with the instantiation described in "5.1 HTTP-based instantiation". As specified by Table I.4 Parameter identifiers in [DASH-AMD3], the DASH Client shall insert in the query string "the latest received value of the header-name HTTP header in the HTTP responses indicated by the `@headerParamSource` attribute".

This way, an Authorization Server located on the Segment Server may send a refreshed Access Token upon reception of a segment request containing a valid Access Token. See "7.4. Subsequent Signed Token Generation" of [URISigning-HAS] for more details on refreshing Access Tokens.

5.4.1 With dynamic MPDs

When MPD update mechanism is used, the DASH client regularly sends an HTTP request at the MPD location. In addition, it is possible to protect the MPD and the segments using the same Access Token using an appropriate path pattern value (see URI Pattern Container (UPC) in A.1). As a result, it is possible to refresh the Access Token every time the DASH Client requests a new MPD, hence preventing the Access Token to expire if for instance the playback is paused for a certain amount of time. To this end, the validity period of the Access Token should be set to a value at least equal to frequency at which DASH Clients fetch MPD updates.

A Annex A – Overview of the Signed Token (informative)

The following gives an overview of relevant information related to the Signed Token of [URISigning] and [URISigning-HAS]. However, it is encouraged to read the original specifications, especially when completeness is necessary such as for implementations.

The URI Signing specification [URISigning] defines the Signed Token format composed of three types of information elements:

- *Enforcement Information Elements*: Information Elements that are used to enforce a distribution policy defined by the CSP. Examples of enforcement attributes are IP address of the UA and time window.
- *Signature Computation Information Elements*: Information Elements that are used by the CDN to verify the URI signature embedded in the received URI. In order to verify a URI Signature, the CDN requires some information elements that describe how the URI Signature was generated. Examples of Signature Computation Elements include the used HMACs hash function and/or the key identifier.
- *URI Signature Information Elements*: The information elements that carry the actual message digest or digital signature representing the URI signature used for checking the integrity and authenticity of the URI. A typical Signed URI will only contain one embedded URI Signature Information Element.

A.1 Enforcement Information Elements

The following parameters determines whether the requested resource may be delivered to the client:

- *Expiry Time (ET) [optional]*: Time when the Signed URI expires. The request is rejected if the received time is later than this timestamp.
- *Client IP (CIP) [optional]*: IP address, or IP prefix, for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 or canonical text representation for IPv6 addresses [RFC5952]. The request is rejected if sourced from a client outside of the specified IP range.
- *Original URI Container (OUC) [optional]*: Container for holding the Full Original URI while the URI signature is calculated. The Original URI Container information element is not transmitted as part of the URI Signing Package Attribute. If the Original URI Container information element is used, the URI Pattern Sequence information element MUST NOT be used.
- *URI Pattern Container (UPC) [optional]*: Container for one or more URI Patterns that describes for which content the Signed URI is valid. The URI Pattern Container contains an expression to match against the requested URI to check whether the requested content is allowed to be requested. Multiple URI Patterns may be concatenated in a single URI Pattern Container information element by separating them with a semi-colon (';') character. If the UPC is used, the Original URI Container information element MUST NOT be used.

See "2.1. Enforcement Information Elements" in [URISigning] for further details on each parameter.

A.2 Signature Computation Information Elements

The [URISigning] specification defines signature computation Information Elements in "2.2. Signature Computation Information Elements". In addition, the [URISigning-HAS] specification extends this set of parameters with the following parameters:

- *Expiration Time Setting (ETS) [optional]*: An 16-bit unsigned integer (in seconds) used for setting the value of the Expiry Time Information Element in newly generated Signed Tokens.
- *Signed Token Transport (STT) [mandatory]*: An 8-bit unsigned integer used for signalling the method through which the Signed Token is transported from the CDN to the UA and vice versa. This document only defines setting the STT Information Element to a value of 1, which means that the Signed Token is transported via a Cookie for both directions.

See "3.1. Signature Computation Information Elements" in [URISigning-HAS] for further details.

A.3 URI Signature Information Elements

See [URISigning-HAS] for further details.

A.4 Signed Token example

The following string is a valid Signed Token:

VER=2&
ET=1209422976&
ETS=15&
CIP=192.0.2.1&
UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&
KID=example:keys:123&
DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E

Figure 6 - Example Signed Token

B Annex B - External AA protocols for Access Token instantiation (informative)

There are possible AA protocols that can be used as external protocols. However, their integration has been verified and may require non-interoperable measure to implement them.

There protocols are:

- Multimedia Authorization Protocol (OMAP)
- The OAuth 2.0 Authorization Framework [OAuth]
- EBU's Cross-Platform Authentication [CPA]

C Annex C - Overview of Signaling and Exchange Mechanisms in MPEG DASH (informative)

[Editor's note: This annex is only for convenience during the review period and may be removed before publication]

C.1 Introduction

This section provides an overview of the signaling and exchange mechanism in [DASH-AMD3].

C.2 Extended UrlQueryInfo in AMD 3

The Amendment 3 [DASH-AMD3] introduces the **`ExtUrlQueryInfo`** element. It exhibits features to support advanced workflows desirable for the exchange of AA Tokens. For instance, the MPD author can indicate that the value of a query string parameter of a segment request is to be found in headers of HTTP responses. In addition, the type of HTTP responses to be inspected for values can also be explicitly signaled, namely "segment", "xlink", "mpd" and "callback". This typically enables the retrieval of AA Tokens in HTTP responses headers and its insertion as query string parameters in future segment requests by the DASH Client.

Here is an example of implementing a token exchange between server and DASH Client. Let us assume that the CDN provides an access token in the HTTP header named "AA-token" in a MPD response, the following MPD example instructs the DASH client to extract the value of this header from MPD and segment responses and to insert it back in the query string parameter "AA-token" for MPD and segment requests.

In particular, the DASH client parses the `@queryTemplate` attribute of the **`ExtUrlQueryInfo`** element. The header key (left to ":") indicates that the values have to be extracted from headers of HTTP responses. The header name (right to ":") indicates the name of the header whose values needs to be extracted by the DASH client. In this example, the DASH client will extract the value of the HTTP header "AA-token-server" from MPD responses (see `@headerParamSource` attribute) and will insert it in every segment and MPD request (see `@includeInRequests`) in the query string "AA-token".

```
<AdaptationSet mimeType="video/mp4" segmentAlignment="true" startWithSAP="1" maxWidth="1280"
maxHeight="720" maxFrameRate="25" par="16:9">
  <EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2016:querystring"
xmlns:up="urn:mpeg:dash:schema:urlparam:2016">
    <up:ExtUrlQueryInfo
        headerParamSource="mpd"
        includeInRequests="segment mpd"
        queryTemplate="AA-token=$header:AA-token-server$"/>
  </EssentialProperty>
  <SegmentTemplate duration="2" startNumber="1" media="video $Number$ $Bandwidth$bps.mp4">
  </SegmentTemplate>
    <Representation id="v0" codecs="avc3.4d401f" width="1280" height="720" frameRate="25"
sar="1:1" bandwidth="3000000"/>
    <Representation id="v1" codecs="avc3.4d401f" width="640" height="360" frameRate="25"
sar="1:1" bandwidth="1500000"/>
</AdaptationSet>
```

Let us assume that the HTTP response for the MPD is as follows:

```
HTTP/1.1 200 OK
Content-Length: 3458
Cache-Control: max-age=86400
```

```
Content-Type: application/dash+xml
AA-token-server: abcdef

<?xml version="1.0" encoding="UTF-8"?>
<MPD>
…
</MPD>
```

1. Computation of an initial query string:

   initialQueryString = "AA-token-server=abcdef"

2. Computation of a final query string:

   finalQueryString = "AA-token=abcdef"

3. Modified media segment URLs building process:

`http://www.example.com/dash/video_1_3000000bps.mp4?AA-token=abcdef`

`http://www.example.com/dash/video_2_3000000bps.mp4?AA-token=abcdef`

`http://www.example.com/dash/video_3_3000000bps.mp4?AA-token=abcdef`

`http://www.example.com/dash/video_4_3000000bps.mp4?AA-token=abcdef`

C.3 Client Authentication and Content Authorization in AMD 3

```
<EssentialProperty  schemeIdUri="urn:org:example:plan-a"
  id="mpeg:dash:client-authentication:2014"
  value="http://authentication.serverA.com/protocolA?=ServiceSpecificInfoA"/>

<EssentialProperty  schemeIdUri="urn:org:example:plan-b"
  id="mpeg:dash:client-authentication:2014"
  value="http:// authentication.serverA.com/protocolB?=ServiceSpecificInfoB"/>

<EssentialProperty  schemeIdUri="urn:org:example:plan-c"
  id="mpeg:dash:content-authorization:2014"
  value="http://authorization.serverC.com/protocolC?=ContentSpecificInfoC"/>

<EssentialProperty  schemeIdUri="urn:org:example:plan-d"
  id="mpeg:dash:content-authorization:2014"
  value="http://authorization.serverD.com/protocolD?=ContentSpecificInfoD"/>
```

The following examples illustrate how client authentication and content access authorization information is signaled in the MPD according to [DASH-AMD3].

The `@id` attribute have specific values that indicate the type of the scheme, namely authentication or authorization with respectively, `mpeg:dash:client-authentication:2014` and `mpeg:dash:client-authorization:2014`.