CDNI Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 21, 2019

F. Fieau, Ed.
E. Stephan
Orange
S. Mishra
Verizon

September 17, 2018

CDNI extensions for HTTPS delegation
draft-fieau-cdni-interfaces-https-delegation-05

Abstract

The delivery of content over HTTPS involving multiple CDNs raises credential management issues. This document proposes extensions in CDNI Control and Metadata interfaces to setup HTTPS delegation from an Upstream CDN (uCDN) to a Downstream CDN (dCDN).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 21, 2019.

Table of Contents

1. Introduction

Content delivery over HTTPS using one or more CDNs along the path requires credential management. This specifically applies when an entity delegates delivery of encrypted content to another trusted entity.

Several delegation methods are currently proposed within different IETF working groups. They specify different methods for provisioning HTTPS delivery credentials.

This document extends the CDNI Metadata interface to setup HTTPS delegation between an upstream CDN (uCDN) and downstream CDN (dCDN). Furthermore, it includes a proposal of IANA registry to enable the adding of new methods.

Section 2 is about terminology used in this document. Section 3 presents delegation methods specified at the IETF. Section 4 addresses the extension for handling HTTPS delegation in CDNI. Section 5 describes simple data types. Section 6 is about an IANA registry for delegation methods. Section 7 raises the security issues.

2. Terminology

This document uses terminology from CDNI framework documents such as: CDNI framework document [RFC7336], CDNI requirements [RFC7337] and CDNI interface specifications documents: CDNI Metadata interface [RFC8006] and CDNI Control interface / Triggers [RFC8007].

3. Known delegation methods

There are currently two Internet drafts within the TLS and ACME working groups adopted to handle delegation of HTTPS delivery between entities.

This I-D proposes standardizing HTTPS delegation between the entities using CDNI interfaces.

This document considers the following two I-D that supports HTTPS delegation:

- Sub-certificates [I-D.ietf-tls-subcerts]

- Short-term certificates in ACME using STAR API [I-D.ietf-acme-star]

4. Extending the CDNI metadata model

This section defines a CDNI extension to the current Metadata interface model that allows bootstrapping delegation methods between a uCDN and a delegate dCDN.

4.1. Extension to PathMetadata object

This extension reuses PathMetadata object, as defined in [RFC8006], by adding new "Delegation methods" objects as specified in the following sections.

This allows to explicitly indicate support for the given method. Therefore, the presence (or lack thereof) of an AcmeStarDelegationMethod, SubcertsDelegationMethod, and/or further delegation methods, imply support (or lack thereof) for the given method.

Example:

The PathMatch object can reference a path-metadata that points at the delegation information. Delegation metadata are added to PathMetaData object.

```
PathMatch:
{
 "path-pattern": {
      "pattern": "/movies/*",
      "case-sensitive": true
 },
 "path-metadata": {
   "type": "MI.PathMetadata",
     "href": "https://metadata.ucdn.example/video.example.com/movies"
 }
}
```

Below shows the PathMetaData Object related to /movie/*
(located at https://metadata.ucdn.example/video.example.com/movies)

```
PathMetadata:
{
    "metadata": [
     {
      "generic-metadata-type": "MI.TimeWindowACL",
      "generic-metadata-value": {
      "times": [{
       "windows": [
       {
        "start": "1213948800",
        "end": "1478047392"
       }]}],
      "action": "allow",
     }},
     {
        "generic-metadata-type": "MI.AcmeStarDelegationMethod",
        "generic-metadata-value": {
        "starproxy": "10.2.2.2",
        "acmeserver" : "10.2.3.3",
        "credentialslocationuri": "www.ucdn.com/credentials",
        "periodicity": 36000
     }}]
}
```

The existence of the "MI.AcmeStarDelegationMethod" object in a PathMetaData Object shall enable the use of one of the AcmeStarDelegation Methods, chosen by the delegate. The delegation method will be activated for the set of Path defined in the PathMatch. See next section for more details about delegation methods metadata specification.

4.2. Delegation methods

This section defines the delegation methods objects metadata. Those metadata allows bootstrapping a secured delegatioin by providing the dCDN with the needed parameters to set it up.

4.2.1. AcmeStarDelegationMethod object

This section defines the AcmeStarDelegationMethod object which describes metadata related to the use of Acme Star API presented in [I-D.ietf-acme-star]

As expressed in [I-D.ietf-acme-star], when an origin has set a delegation to a specific domain (i.e. dCDN), the dCDN should present to the end-user client, a short-term certificate bound to the master certificate.

Property: starproxy

Description: Used to advertise the STAR Proxy to the dCDN. Endpoint type defined in RFC8006, section 4.3.3

Type: Endpoint

Mandatory-to-Specify: Yes

Property: acmeserver

Description: used to advertise the ACME server to the dCDN. Endpoint type is defined in RFC8006, section 4.3.3

Type: Endpoint

Mandatory-to-Specify: Yes

Property: credentialslocationuri

Description: expresses the location of the credentials to be fetched by the dCDN. Link type is as defined in RFC8006, section 4.3.1

Type: Link

Mandatory-to-Specify: Yes

Property: periodicity

Description: expresses the credentials renewal periodicity. See next section on simple meta data type.

Type: Periodicity

Mandatory-to-Specify: Yes

As an example, AcmeStarDelegationMethod object could express the Acme-Star delegation as the following:

```
AcmeStarDelegationMethod: {
    "generic-metadata-type": "MI.AcmeStarDelegationMethod",
    "generic-metadata-value": {
        "starproxy": "10.2.2.2",
        "acmeserver": "10.2.3.3",
        "credentialslocationuri": "www.ucdn.com/credentials",
        "periodicity": 36000
    }
}
```

4.2.2. SubcertsDelegationMethod object

This section defines the SubcertsDelegationMethod object which describes metadata related to the use of Subcerts as presented in [I-D.ietf-tls-subcerts]

As expressed in [I-D.ietf-tls-subcerts], when an origin has set a delegation to a specific domain (i.e. dCDN), the dCDN should present the Origin or uCDN certificate or "delegated_credential" during the TLS handshake [RFC8446] to the end-user client application, instead of its own certificate.

Property: credentialsdelegatingentity

Description: Endpoint ID (IP) of the delegating Entity (uCDN). Endpoint type defined in RFC8006, section 4.3.3

Type: Endpoint

Mandatory-to-Specify: Yes

Property: credentialrecipiententity

Description: Endpoint ID (IP) of the delegated entity (dCDN). Endpoint type is defined in RFC8006, section 4.3.3

Type: Endpoint

Mandatory-to-Specify: Yes

Property: credentialslocationuri

Description: expresses the location of the credentials to be fetched by the dCDN. Link type is as defined in RFC8006, section 4.3.1

Type: Link

Mandatory-to-Specify: Yes

Property: periodicity

Description: expresses the credentials renewal periodicity. See next section on simple meta data type.

Type: Periodicity

Mandatory-to-Specify: Yes

As an example, when a uCDN has delegated HTTPS delivery to dCDN, a SubcertsDelegationMethod object can express the SubCerts delegation as the following:

```
SubcertsDelegationMethod: {
    "generic-metadata-type": "MI.SubcertsDelegationMethod",
    "generic-metadata-value": {
        "credentialsdelegatingentity": "10.2.2.2",
        "credentialsrecepiententity": "10.2.3.3",
        "credentialslocationuri": "www.ucdn.com/credentials",
        "periodicity": 36000
    }
}
```

5. Metadata Simple Data Type Descriptions

This section describes the simple data types that are used for properties for objects in this document.

5.1. Periodicity

A time value expressed in seconds to indicate a periodicity.

Type: Integer

6. IANA considerations

This document requests the registration of the following entries under the "CDNI Payload Types" registry hosted by IANA regarding "CDNI delegation":

Payload Type	Specification
MI.AcmeStarDelegationMethod	RFCthis
MI.SubCertDelegationMethod	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.1. CDNI MI AcmeStarDelegationMethod Payload Type

Purpose: The purpose of this Payload Type is to distinguish AcmeStarDelegationMethod MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.1

6.2. CDNI MI SubCertsDelegationMethod Payload Type

Purpose: The purpose of this Payload Type is to distinguish SubcertsDelegationMethod MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 4.2.2

7. Security considerations

Extensions proposed here do not change Security Considerations as outlined in the CDNI Metadata and Footprint and Capabilities RFCs [RFC8006].

8. References

8.1. Normative References

[RFC7937] Le Faucheur, F., Ed., Bertrand, G., Ed., Oprescu, I., Ed., and R. Peterkofsky, "Content Distribution Network Interconnection (CDNI) Logging Interface", RFC 7937, DOI 10.17487/RFC7937, August 2016, <https://www.rfc-editor.org/info/rfc7937>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/info/rfc8007>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

8.2. Informative References

[I-D.ietf-acme-star] Sheffer, Y., Lopez, D., Dios, O., Pastor, A., and T. Fossati, "Support for Short-Term, Automatically-Renewed (STAR) Certificates in Automated Certificate Management Environment (ACME)", draft-ietf-acme-star-03 (work in progress), March 2018.

[I-D.ietf-tls-subcerts] Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS", draft-ietf-tls-subcerts-02 (work in progress), August 2018.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <https://www.rfc-editor.org/info/rfc7337>.

Authors' Addresses

Frederic Fieau (editor)
Orange
40-48, avenue de la Republique
Chatillon 92320
France

Email: frederic.fieau@orange.com

Emile Stephan
Orange
2, avenue Pierre Marzin
Lannion 22300
France

Email: emile.stephan@orange.com

Sanjay Mishra
Verizon
13100 Columbia Pike
Silver Spring MD 20904
USA

Email: sanjay.mishra@verizon.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2019

O. Finkelman
Qwilt
S. Mishra
Verizon

October 22, 2018

CDNI Control Triggers Interface Extensions
draft-finkelman-cdni-triggers-sva-extensions-01

Abstract

The Open Caching working group of the Streaming Video Alliance is focused on the delegation of video delivery request from commercial CDNs to a caching layer at the ISP. In that aspect, Open Caching is a specific use case of CDNI, where the commercial CDN is the upstream CDN (uCDN) and the ISP caching layer is the downstream CDN (dCDN). The extensions specified in this document to the CDNI CI/T interface are derived from requirements raised by Open Caching but are applicable to CDNI use cases in general.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2019.

Table of Contents

1. Introduction

This document defines the objects and extensions required for granular content management operations. For that purpose it extends CDNI Control Interface/Triggers [RFC8007]. The basic operations are the ones defined in the RFC (i.e. purge, invalidate, pre-position). For consistency, this document follows the CDNI notation of uCDN (the commercial CDN) and dCDN (the ISP caching layer). When using the term CP in this document we refer to a video content provider.

The CDNI metadata interface is described in [RFC8006].

The CDNI footprint and capability interface is described in [RFC8008].

The CDNI control interface / triggers is described in [RFC8007].

1.1. Terminology

This document reuses the terminology defined in [RFC6707], [RFC8006], [RFC8007], and [RFC8008].

Additionally, the following terms are used throughout this document and are defined as follows:

- HLS - HTTP Live Streaming

- DASH – Dynamic Adaptive Streaming Over HTTP
- MSS – Microsoft Smooth Streaming

2. Interfaces Extensions Overview

This document defines extensions for the CDNI Control Interface / Triggers [RFC8007] and defines FCI objects as per the CDNI Footprint and Capabilities Interface [RFC8008].

2.1. CDNI Control Interface / Triggers Extensions

2.1.1. CI/T Objects

This document specifies version 2 of the CI/T objects in order to support version 2 of the Trigger Specification as required below in Section 2.1.2.

2.1.2. Trigger Specification

This document specifies version 2 of the Trigger Specification which is an enhancement of the Trigger Specification that includes all properties as defined in section 5.2.1 of [RFC8007] as well as the additional properties required by the use cases listed below in Section 2.1.3 and Section 2.1.4.

2.1.3. Content Selection

The trigger specification as defined in section 5.2.1 of [RFC8007] provides means to select content objects by matching a full content URL or patterns with wildcards. This document specifies two additional selection options.

- Regular Expression – Using regex a uCDN can create more complex rules to select the content objects for the cases of invalidation and purge. For example, purging specific content within a specific directory path.
- Content Playlist – Using video playlist files, a uCDN can trigger an operation that will be applied to a collection of distinct media files in a format that is natural for a streaming video content provider. A playlist may have several formats, specifically HTTP Live Streaming (HLS) *.m3u8 manifest [RFC8216], Microsoft Smooth Streaming (MSS) *.ismc client manifest [MSS], and Dynamic Adaptive Streaming over HTTP (DASH) *.mpd file [ISO/IEC 23009-1:2014] [MPEG-DASH].

2.1.4. Trigger Extensibility

The CDNI Control Interface / Triggers [RFC8007] defines a set of objects used by the trigger commands. In order to have better control and finer granularity, we define a mechanism for generic trigger extension object wrapper for managing individual CDNI trigger extensions in an opaque manner, as well as an initial set of trigger extension objects.

This document also registers CDNI Payload Types [RFC7736] under the namespace CIT for the initial set of trigger extension types:

- CIT.LocationPolicy (for controlling the locations in which the trigger is executed)
- CIT.TimePolicy (for scheduling a trigger to run in a specific time window)

Example use cases

- Pre-position with cache location policy
- Purge content with cache location policy
- Pre-position at a specific time
- Purge by content acquisition time (e.g. purge all content acquired in the past X hours)

2.1.5. Error Propagation

As triggers may be propagated over a chain of downstream CDNs and since, in some cases, triggers may be redistributed from dCDN-A to dCDN-B even if dCDN-A does not understand a specific extension, it is essential for the uCDN that sets the trigger to be able to trace back and error to the downstream where it occurred. This document specifies version 2 of the Error Description which is an enhancement of the Error Description as defined in section 5.2.6 of [RFC8007] and that includes all the original properties as well as the additional property "cdn" which is an identifier for the faulty CDN. When a downstream dCDN-A propagates a trigger to another downstream dCDN-B, it MUST also propagate back the errors received in the trigger status resource from dCDN-B. This makes sure that the trigger originating upstream CDN will receive an array of errors that occurred in all the CDNs along the execution path, each error carrying its own CDN identifier.

2.2. CDNI Footprint and Capabilities Interface Extensions

Extending the trigger mechanism with optional properties requires the ability for the dCDN to advertise which optional properties it supports.

The CDNI Footprint and Capabilities Interface [RFC8008] enables the dCDN to advertise the capabilities it supports across different footprints. This document introduces FCI objects to support the advertisement of these optional properties.

Example use cases

- Trigger types: Advertise which trigger types are supported by the dCDN. CDNI defines three trigger types (purge, invalidate, pre-position), but it does not necessarily mean that all dCDNs support all of them. The uCDN may prefer to work only with dCDN that support what the uCDN needs.

- Content selection rule types: Advertise which selection types are supported. For example, if adding content regex as a means to match on content URLs, not all dCDN would support it. For playlist mapping, advertise which types and versions of protocols are supported, e.g. HLS.vX/DASH.vY/MSS.vX, DASH templates. Note that the version string or schema are protocol specific.

- Trigger extensions: Advertise which trigger extensions object types are supported by the dCDN.

3. CI/T Version 2

[RFC8007] does not define a version number and versioning scheme. We, therefore, designate the interface and objects as defined in section 5 of [RFC8007] as version 1. The following sections define version 2 of the CI/T objects and their properties as extensions of version 1.

3.1. CI/T Objects V2

Version 2 of the CI/T interface requires the support of the following objects:

- CI/T Commands v2: A trigger command request using the payload type ci-trigger-command.v2. Version 2 MUST only use "trigger.v2" objects as defined in Section 3.2.1, instead of "trigger" objects. All other properties of the trigger command v2 are as defined in section 5.1.1 of [RFC8007].

- Trigger Status Resource v2: A trigger status resource response using the payload type ci-trigger-status.v2. Version 2 MUST only use "trigger.v2" objects as defined in Section 3.2.1, instead of a "trigger" object, as well as "errors.v2" objects as defined in Section 3.2.6, instead of a "errors" object All other properties of the trigger status v2 are as defined in section 5.1.2 of [RFC8007]. The errors array "errors.v2" is a list of all errors that occurred in any of the downstream CDNs along the execution path. When a downstream CDN, dCDN-A, propagates a trigger to another downstream CDN, dCDN-B, it MUST also propagated back all errors reported by dCDN-B in the trigger status resource and add them to its own trigger status resource.

- Trigger Collections: The payload type ci-trigger-collection is used with no changes and as defined in 5.1.3 of [RFC8007].

Usage example of version 2 of trigger command

REQUEST:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: triggers.dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command.v2
{
  "trigger.v2": { <properties of a trigger.v2 object> },
  "cdn-path": [ "AS64496:1" ]
}
```

RESPONSE:

```
HTTP/1.1 201 Created
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Length: 467
Content-Type: application/cdni; ptype=ci-trigger-status.v2
Location: https://triggers.dcdn.example.com/triggers/0
Server: example-server/0.1

{
   "errors.v2": [ { <properties of 1st error.v2 object> },
                    ...,
                  { <properties of Nth error.v2 object> }
   ],
   "ctime": 1462351690,
   "etime": 1462351698,
   "mtime": 1462351690,
   "status": "pending",
   "trigger.v2": { <properties of a trigger.v2 object> }
}
```

Usage example of version 2 of trigger status for the trigger created in the above trigger command example:

REQUEST:

```
GET /triggers/0 HTTP/1.1
User-Agent: example-user-agent/0.1
Host: triggers.dcdn.example.com
Accept: */*
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Length: 467
Expires: Wed, 04 May 2016 08:49:10 GMT
Server: example-server/0.1
ETag: "6990548174277557683"
Cache-Control: max-age=60
Date: Wed, 04 May 2016 08:48:10 GMT
Content-Type: application/cdni; ptype=ci-trigger-status.v2

{
   "errors.v2": [ { <properties of 1st error.v2 object> },
                  ...,
                { <properties of Nth error.v2 object> }
   ],
   "ctime": 1462351690,
   "etime": 1462351698,
   "mtime": 1462351690,
   "status": "pending",
   "trigger.v2": { <properties of a trigger.v2 object> }
}
```

3.2. Properties of CI/T Version 2 objects

This section defines the values that can appear in the top-level objects described in Section 3.1, and their encodings.

3.2.1. Trigger Specification Version 2

Version 2 of the Trigger Specification adds the following properties on top of the existing properties of the trigger specification defined in section 5.2.1 of [RFC8007].

Property: content.regexs

Description: Regexs of content URLs to which the CI/T trigger command applies.

Type: A JSON array of RegexMatch objects (see Section 3.2.2).

Mandatory: No, but at least one of "metadata.*" or "content.*" MUST be present and non-empty.

Property: content.playlists

Description: Playlists of content the CI/T trigger command applies to.

Type: A JSON array of Playlist objects (see Section 3.2.3).

Mandatory: No, but at least one of "metadata.*" or "content.*" MUST be present and non-empty.

Property: extensions

Description: Array of trigger extension data.

Type: Array of GenericTriggerExtension objects (see Section 3.2.5.2).

Mandatory-to-Specify: No. The default is no extensions.

Example of an invalidation trigger.v2 with a list of regex objects, a list of playlist objects, and extensions:

```
{
  "trigger.v2": {
    "type": "invalidate",
    "content.regexs": [ <list of RegexMatch objects> ],
    "content.playlists": [ <list of Playlist objects> ],
    "extensions": [ <list of GenericTriggerExtension objects ]
  },
  "cdn-path": [ "AS64496:1" ]
}
```

3.2.2. RegexMatch

A RegexMatch consists of a regular expression string a URI is matched against, and flags describing the type of match. It is encoded as a JSON object with following properties:

Property: regex

Description: A regular expression for URI matching.

Type: A regular expression to match against the URI, i.e against the path-absolute and the query string parameters

[RFC3986]. The regular expression string MUST be compatible with PCRE [PCRE841].

Note: Because '\' has special meaning in JSON [RFC8259] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

Mandatory: Yes.

Property: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used.

Type: JSON boolean. Either "true" (the matching is case sensitive) or "false" (the matching is case insensitive).

Mandatory: No; default is case-insensitive match (i.e., a value of "false").

Property: match-query-string

Description: Flag indicating whether to include the query part of the URI when comparing against the regex.

Type: JSON boolean. Either "true" (the full URI, including the query part, should be compared against the regex) or "false" (the query part of the URI should be dropped before comparison with the given regex).

Mandatory: No; default is "false". The query part of the URI MUST be dropped before comparison with the given regex. This makes the regular expression simpler and safer for cases in which the query parameters are not relevant for the match.

Example of a case sensitive, no query parameters, regex match against:
"^(https:\/\/video\.example\.com)\/([a-z])\/movie1\/([1-7])\/*(index.m3u8|\d{3}.ts)$".

This regex matches URLs of domain video.example.com where the path structure is /(single lower case letter)/(name-of-title)/(single digit between 1 to 7)/(index.m3u8 or a 3 digit number with ts extension). For example: https://video.example.com/d/movie1/5/index.m3u8 or https://video.example.com/k/movie1/4/013.ts.

```
{
 "regex": "^(https:\\/\\/video\\.example\\.com)\\/([a-z])\\/movie1\
           \/([1-7])\\/*(index.m3u8|\\d{3}.ts)$",
 "case-sensitive": true,
 "match-query-string": false
}
```

3.2.3. Playlist

A Playlist consists of a full URL and a media protocol identifier. An implementation that supports a specific playlist media protocol MUST be able to parse playlist files of that protocol type and extract, possibly recursively, the URLs to all media objects and/or sub playlist files, and apply the trigger to each one of them separately.

Playlist is encoded as a JSON object with following properties:

Property: playlist

Description: A URL to the playlist file.

Type: A URL represented as a JSON string.

Mandatory: Yes.

Property: media-protocol

Description: Media protocol to be when parsing and interpreting this playlist.

Type: MediaProtocol (see Section 3.2.4).

Mandatory: Yes.

Example of a HLS playlist:

```
{
 "playlist": "https://www.example.com/hls/title/index.m3u8",
 "media-protocol": "hls"
}
```

3.2.4. MediaProtocol

Media Protocol objects are used to specify registered type of media protocol (see Section 6.3) used for protocol related operations like pre-position according to playlist.

Type: JSON string

Example:

"dash"

3.2.5. CI/T Trigger Extensions

A "trigger.v2" object, as defined in Section 3.2.1 includes an optional array of trigger extension objects. A trigger extension contain properties that are used as directives for dCDN when executing the trigger command -- for example, location policies, time policies and so on. Each such CDNI Trigger extension is a specialization of a CDNI GenericTriggerExtension object. The GenericTriggerExtension object abstracts the basic information required for trigger distribution from the specifics of any given property (i.e., property semantics, enforcement options, etc.). All trigger extensions are optional, and it is thus the responsibility of the extension specification to define a consistent default behavior for the case the extension is not present.

3.2.5.1. Enforcement Options

The trigger enforcement options concept is in accordance with the metadata enforcement options as defined in section 3.2 of [RFC8006].

The GenericTriggerExtension object defines the properties contained within it as well as whether or not the properties are "mandatory-to-enforce". If the dCDN does not understand or support a mandatory-to-enforce property, the dCDN MUST NOT execute the trigger command. If the extension is not mandatory-to-enforce, then that GenericTriggerExtension object can be safely ignored and the trigger command can be processed in accordance with the rest of the CDNI Trigger spec.

Although a CDN MUST NOT execute a trigger command if a mandatory-to-enforce extension cannot be enforced, it could still be safe to redistribute that trigger (the "safe-to-redistribute" property) to another CDN without modification. For example, in the cascaded CDN case, a transit CDN (tCDN) could convey mandatory-to-enforce trigger extension to a dCDN. For a trigger extension that does not require customization or translation (i.e., trigger extension that is safe-

to-redistribute), the data representation received off the wire MAY be stored and redistributed without being understood or supported by the tCDN. However, for trigger extension that requires translation, transparent redistribution of the uCDN trigger values might not be appropriate. Certain triggers extensions can be safely, though perhaps not optimally, redistributed unmodified. For example, pre-position command might be executed in suboptimal times for some geographies if transparently redistributed, but it might still work.

Redistribution safety MUST be specified for each GenericTriggerExtension property. If a CDN does not understand or support a given GenericTriggerExtension property that is not safe-to-redistribute, the CDN MUST set the "incomprehensible" flag to true for that GenericTriggerExtension object before redistributing it. The "incomprehensible" flag signals to a dCDN that trigger metadata was not properly transformed by the tCDN. A CDN MUST NOT attempt to execute a trigger that has been marked as "incomprehensible" by a uCDN.

tCDNs MUST NOT change the value of mandatory-to-enforce or safe-to-redistribute when propagating a trigger to a dCDN. Although a tCDN can set the value of "incomprehensible" to true, a tCDN MUST NOT change the value of "incomprehensible" from true to false.

Table 1 describes the action to be taken by a tCDN for the different combinations of mandatory-to-enforce ("MtE") and safe-to-redistribute ("StR") properties when the tCDN either does or does not understand the trigger extension object in question:

MtE	StR	Extension object understood by tCDN	Trigger action
False	True	True	Can execute and redistribute.
False	True	False	Can execute and redistribute.
False	False	False	Can execute. MUST set "incomprehensible" to true when redistributing.
False	False	True	Can execute. Can redistribute after transforming the trigger extension (if the CDN knows how to do so safely); otherwise, MUST set "incomprehensible" to true when redistributing.
True	True	True	Can execute and redistribute.
True	True	False	MUST NOT execute but can redistribute..
True	False	True	Can execute. Can redistribute after transforming the trigger extension (if the CDN knows how to do so safely); otherwise, MUST set "incomprehensible" to true when redistributing.
True	False	False	MUST NOT serve. MUST set "incomprehensible" to true when redistributing.

Table 1: Action to be taken by a tCDN for the different combinations of MtE and StR properties

Table 2 describes the action to be taken by a tCDN for the different combinations of mandatory-to-enforce and "incomprehensible" ("Incomp") properties, when the dCDN either does or does not understand the trigger extension object in question:

MtE	Incomp	Extension object understood by dCDN	Trigger action
False	False	True	Can execute.
False	True	True	Can execute but MUST NOT interpret/apply any trigger extension marked as "incomprehensible".
False	False	False	Can execute.
False	True	False	Can execute but MUST NOT interpret/apply any trigger extension marked as "incomprehensible".
True	False	True	Can execute.
True	True	True	MUST NOT execute.
True	False	False	MUST NOT execute.
True	True	False	MUST NOT execute.

Table 2: Action to be taken by a dCDN for the different combinations of MtE and Incomp properties

3.2.5.2. GenericExtensionObject

A GenericTriggerExtension object is a wrapper for managing individual CDNI Trigger extensions in an opaque manner.

Property: generic-trigger-extension-type

Description: Case-insensitive CDNI Trigger extension object type.

Type: String containing the CDNI Payload Type [RFC7736] of the object contained in the generic-trigger-extension-value property (see table in Section 6.1).

Mandatory-to-Specify: Yes.

Property: generic-trigger-extension-value

Description: CDNI Trigger extension object.

Type: Format/Type is defined by the value of the generic-trigger-extension-type property above.

Mandatory-to-Specify: Yes.

Property: mandatory-to-enforce

Description: Flag identifying whether or not the enforcement of this trigger extension is mandatory.

Type: Boolean

Mandatory-to-Specify: No. Default is to treat the trigger extension as mandatory-to-enforce (i.e., a value of True).

Property: safe-to-redistribute

Description: Flag identifying whether or not this trigger extension can be safely redistributed without modification.

Type: Boolean

Mandatory-to-Specify: No. Default is to allow transparent redistribution (i.e., a value of True).

Property: incomprehensible

Description: Flag identifying whether or not any CDN in the chain of delegation has failed to understand and/or failed to properly transform this trigger extension object. Note: This flag only applies to trigger extension objects whose safe-to-redistribute property has a value of False.

Type: Boolean

Mandatory-to-Specify: No. Default is comprehensible (i.e., a value of False).

Example of a GenericTriggerExtension containing a specific trigger extension object:

```
{
  "generic-trigger-extension-type":
    <Type of this trigger extension object>,
  "generic-trigger-extension-value":
    {
      <properties of this trigger extension object>
    },
  "mandatory-to-enforce": true,
  "safe-to-redistribute": true,
  "incomprehensible": false
}
```

3.2.6. Error Description Version 2

Version 2 of the Error Description adds the "cdn" property on top of the existing properties of the trigger Error Description as defined in section 5.2.6 of [RFC8007]. The "cdn" property identifies the CDN in which the error have occurred.

Property: cdn

Description: The CDN PID of the CDN where the error occurred.

Type: A non-empty JSON string, where the string is a CDN PID as defined in section 4.6 of [RFC8007].

Mandatory: Yes.

Example of an errors.v2 with a an error of unsupported location policy extension object:

```
{
  "errors.v2": [
   {
     "content.urls": [
        "https://newsite.example.com/index.html"
      ],
     "description": "unrecoginzed extension type CIT.LocationPolicy",
     "error": "eunsupported",
     "cdn": "AS64496:1"
   },
  ]
}
```

3.2.7. Error codes

This document adds the error code "eextension" to the error codes table defined in section 5.2.6 of [RFC8007]. This error code designates that an error occurred while parsing a generic trigger extension, or that the specific extension is not supported by the CDN. A CDN that fails to parse or execute a generic extension object MUST report it using the "errors.v2" array within the trigger status resource, while setting the error code to "eextension" and providing an appropriate description. The "eextension" error code is a registered type of "CDNI CI/T Trigger Error Codes" (see Section 6.2).

4. Trigger Extension Objects

The objects defined below are intended to be used in the GenericTriggerExtension object's generic-trigger-extension-value field as defined in section Section 3.2.5.2, and their generic-trigger-extension-type property MUST be set to the appropriate CDNI Payload Type as defined in Section 6.1 .

4.1. LocationPolicy extension

A content operation may be relevant for a specific geographical region, or need to be excluded from a specific region. In this case, the trigger should be applied only to parts of the network that are either "included" or "not excluded" by the location policy. Note that the restrictions here are on the cache location rather than the client location.

The LocationPolicy object defines which CDN or cache locations for which the trigger command is relevant.

Example use cases:

- Pre-position: Certain contracts allow for pre-positioning or availability of contract in all regions except for certain excluded regions in the world, including caches. For example, some content cannot ever knowingly touch servers in a specific country, including cached content. Therefore, these regions MUST be excluded from a pre-positioning operation.

- Purge: In certain cases, content may have been located on servers in regions where the content must not reside. In such cases a purge operation to remove content specifically from that region, is required.

Object specification

Property: locations

Description: An Access List that allows or denies (blocks) the trigger execution per cache location.

Type: Array of LocationRule objects (see Section 4.2.2.1 of [RFC8006])

Mandatory-to-Specify: Yes.

If a location policy object is not listed within the trigger command, the default behavior is to execute the trigger in all available caches and locations of the dCDN.

The trigger command is allowed, or denied, for a specific cache location according to the action of the first location whose footprint matches against that cache's location. If two or more footprints overlap, the first footprint that matches against the cache's location determines the action a CDN MUST take. If the "locations" property is an empty list or if none of the listed footprints match the location of a specific cache location, then the result is equivalent to a "deny" action.

The following is an example of pre-position trigger specification with a trigger-extensions array including a location policy that allows the trigger execution in the US but blocks its execution in Canada:

```
{
  "trigger": {
    "type": "preposition",
    "content.urls": [
        "https://www.example.com/a/b/c/1",
        "https://www.example.com/a/b/c/2"
    ],
    "extensions": [
       {
          "generic-trigger-extension-type": "CIT.LocationPolicy",
          "generic-trigger-extension-value":
           {
              "locations": [
                {
                  "action": "allow",
                  "footprints": [
                    {
                      "footprint-type": "countrycode",
                      "footprint-value": ["us"]
                    }
                  ]
                },
                {
                  "action": "deny",
                  "footprints": [
                    {
                      "footprint-type": "countrycode",
                      "footprint-value": ["ca"]
                    }
                  ]
                }
              ]
           },
          "mandatory-to-enforce": true,
          "safe-to-redistribute": true,
          "incomprehensible": false
       }
    ]
  },
  "cdn-path": [ "AS64496:1" ]
}
```

4.2. TimePolicy Extension

A uCDN may wish to perform content management operations on the dCDN in a specific schedule. The TimePolicy extensions allows the uCDN to instruct the dCDN to execute the trigger command in a desired time window.

Example use cases

* Pre-position: A content provider wishes to pre-populate a new episode at off-peak time so that it would be ready on caches (for example home caches) at prime time when the episode is released for viewing. A scheduled operation enables the uCDN to direct the dCDN in what time frame to execute the trigger. The time values are in UNIX epoch.

* Regional schedule: When used in combination with the Location Policy defined in Section 4.1, the uCDN can trigger separate commands for different geographical regions, for each region using a different schedule. This allows the uCDN to control the execution time per region and, for example, direct the dCDN to execute at off-peak hours, as they are defined per region.

Object specification

Property: window

Description: A time frame in which the trigger should be executed.

Type: TimeWindow object (see Section 4.2.3.2 of [RFC8006])

Mandatory-to-Specify: Yes.

If a time policy object is not listed within the trigger command, the default behavior is to execute the trigger in a time frame most suitable to the dCDN taking under consideration other constrains and / or obligations.

Example of trigger specification with a scheduled time window between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

```
POST /triggers HTTP/1.1
User-Agent: example-user-agent/0.1
Host: dcdn.example.com
Accept: */*
Content-Type: application/cdni; ptype=ci-trigger-command
Content-Length: 352

{
  "trigger": {
    "type": "preposition",
    "content.urls": [
        "https://www.example.com/a/b/c/1",
        "https://www.example.com/a/b/c/2"
      ],
    "extensions": [
       {
          "generic-trigger-extension-type": "CIT.TimePolicy",
          "generic-trigger-extension-value":
           {
             "window": {
                "start": 946717200,
                "end": 946746000
             }
           }
          "mandatory-to-enforce": true,
          "safe-to-redistribute": true,
          "incomprehensible": false
       }
    ],
  },
  "cdn-path": [ "AS64496:1" ]
}
```

5. Footprint and Capabilities

This section covers the FCI objects required for advertisement of the extensions and properties introduced in this document.

5.1. CI/T Versions Capability Object

The CI/T versions capability object is used to indicate support for one or more CI/T objects versions. Note that the default version as originally defined in [RFC8007] MUST be implicitly supported regardless of the versions listed in this capability object.

Property: versions

Description: A list of version numbers.

Type: An array of JSON strings

Mandatory-to-Specify: No. The default is version 1. A missing or an empty versions list means that only version 1 of the interface and objects is supported.

5.1.1. CI/T Versions Capability Object Serialization

The following shows an example of CI/T Versions Capability object serialization for a dCDN that supports versions 2 and 2.1 of the CI/T interface.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.TriggerVersion",
      "capability-value": {
        "versions": [ "1", "2", "2.1" ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

5.2. CI/T Playlist Protocol Capability Object

The CI/T Playlist Protocol capability object is used to indicate support for one or more MediaProtocols listed in Section 6.3 by the playlists property of the "trigger.v2" object.

Property: media-protocols

Description: A list of media protocols.

Type: A list of MediaProtocol (from the CDNI Triggers media protocol types Section 6.3)

Mandatory-to-Specify: No. The default, in case of a missing or an empty list, is none supported.

5.2.1. CI/T Playlist Protocol Capability Object Serialization

The following shows an example of CI/T Playlist Protocol Capability object serialization for a dCDN that supports "hls" and "dash".

```
{
  "capabilities": [
    {
      "capability-type": "FCI.TriggerPlaylistProtocol",
      "capability-value": {
        "media-protocols": ["hls", "dash"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

5.3. CI/T Trigger Extension Capability Object

The CI/T Generic Extension capability object is used to indicate support for one or more GenericExtensionObject types.

Property: trigger-extension

Description: A list of supported CDNI CI/T GenericExtensionObject types.

Type: List of strings corresponding to entries from the "CDNI Payload Types" registry [RFC7736] that are under the CIT namespace, and that correspond to CDNI CI/T GenericExtensionObject objects.

Mandatory-to-Specify: No. The default, in case of a missing or an empty list, MUST be interpreted as "no GenericExtensionObject types are supported". A non-empty list MUST be interpreted as containing "the only GenericExtensionObject types that are supported".

5.3.1. CI/T Trigger Extension Capability Object Serialization

The following shows an example of CI/T Trigger Extension Capability object serialization for a dCDN that supports the "CIT.LocationPolicy" and the "CIT.TimePolicy" objects.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.TriggerGenericExtension",
      "capability-value": {
        "trigger-extension": ["CIT.LocationPolicy", "CIT.TimePolicy"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

6. IANA Considerations

6.1. CDNI Payload Types

This document requests the registration of the following CDNI Payload Types under the IANA CDNI Payload Type registry defined in [RFC7736]:

Payload Type	Specification
ci-trigger-command.v2	RFCthis
ci-trigger-status.v2	RFCthis
CIT.LocationPolicy	RFCthis
CIT.TimePolicy	RFCthis
FCI.TriggerVersion	RFCthis
FCI.TriggerPlaylistProtocol	RFCthis
FCI.TriggerGenericExtension	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.1.1. CDNI ci-trigger-command.v2 Payload Type

Purpose: The purpose of this payload type is to distinguish version 2 of the CI/T command (and any associated capability advertisement)

Interface: CI/T

Encoding: see Section 3.1

6.1.2. CDNI ci-trigger-status.v2 Payload Type

Purpose: The purpose of this payload type is to distinguish version 2 of the CI/T status resource response (and any associated capability advertisement)

Interface: CI/T

Encoding: see Section 3.1

6.1.3. CDNI CI/T LocationPolicy Trigger Extension Type

Purpose: The purpose of this Trigger Extension type is to distinguish LocationPolicy CIT Trigger Extension objects.

Interface: CI/T

Encoding: see Section 4.1

6.1.4. CDNI CI/T TimePolicy Trigger Extension Type

Purpose: The purpose of this Trigger Extension type is to distinguish TimePolicy CI/T Trigger Extension objects.

Interface: CI/T

Encoding: see Section 4.2

6.1.5. CDNI FCI CI/T Versions Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Triggers Versions objects

Interface: FCI

Encoding: see Section 5.1.1

6.1.6. CDNI FCI CI/T Playlist Protocol Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Playlist Protocol objects

Interface: FCI

Encoding: see Section 5.2.1

6.1.7. CDNI FCI CI/T Extension Objects Payload Type

Purpose: The purpose of this payload type is to distinguish FCI advertisement objects for CI/T Extension objects

Interface: FCI

Encoding: see Section 5.3.1

6.2. CDNI CI/T Trigger Error Codes types

The IANA is requested to update the "CDNI CI/T Error Codes" subregistry (defined in section 7.3 of [RFC8007] and located at <https://www.iana.org/assignments/cdni-parameters>) with the following registration:

Error Code	Description	Specification
eextension	The dCDN failed to parse a generic extension object, or does not support this extension.	Section Section 3.2.7 of this document.

6.3. CDNI Media protocol types

The IANA is requested to create a new "CDNI MediaProtocol Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Media Protocol Types" namespace defines the valid Media Protocol object values in Section Section 3.2.4, used by the Playlist object. Additions to the MediaProtocol namespace conform to the "Specification Required" policy as defined in section 4.6 of [RFC8126], where the specification defines the MediaProtocol Type and the protocol to which it is associated. The designated expert will verify that new protocol definitions do not duplicate existing protocol definitions and prevent gratuitous additions to the namespace.

The following table defines the initial MediaProtocol values corresponding to the HLS, MSS, and DASH protocols:

MediaProtocol Type	Description	Specification	Protocol Specification
hls	HTTP Live Streaming	RFCthis	RFC 8216 [RFC8216]
mss	Microsoft Smooth Streaming	RFCthis	MSS [MSS]
dash	Dynamic Adaptive Streaming over HTTP (MPEG-DASH)	RFCthis	MPEG-DASH [MPEG-DASH]

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

7. Security Considerations

All security considerations listed in section 8 of [RFC8007] and section 7 of [RFC8008] apply to this document as well.

This document defines the capability to use regular expression within the trigger spec for more granular content selection. The usage of regex introduced the risk of regex complexity attacks, a.k.a ReDos attacks. An attacker may be able to craft a regular expression that can exhaust server resources and may take exponential time in in the worst case. An implementation MUST protect itself by at least accept triggers only from an authenticated party over a secured connection. An implementation SHOULD also protect itself by using secure programing techniques and decline trigger commands that use potentially risky regex, such techniques are readily available in secure programming literature and are beyond the scope of this document.

8. Acknowledgments

TBD

9. Contributors

The authors would like to thank all members of the "Streaming Video Alliance" (SVA) Open Caching Working Group for their contribution in support of this document.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/info/rfc8007>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

10.2. Informative References

[MPEG-DASH] ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment format", ISO/IEC 23009-1:2014, Edition 2, 05 2014, <http://www.iso.org/standard/65274.html>.

[MSS] Microsoft, "[MS-SSTR]: Smooth Streaming Protocol", Protocol Revision 8.0, September 2017, <https://msdn.microsoft.com/en-us/library/ff469518.aspx>.

[PCRE841] Hazel, P., "Perl Compatible Regular Expressions", Version 8.41, July 2017, <http://www.pcre.org/>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <https://www.rfc-editor.org/info/rfc7736>.

[RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>.

Authors' Addresses

Ori Finkelman
Qwilt
6, Ha'harash
Hod HaSharon 4524079
Israel

Phone: +972-72-2221647
Email: orif@qwilt.com

Sanjay Mishra
Verizon
13100 Columbia Pike
Silver Spring, MD 20904
USA

Email: sanjay.mishra@verizon.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 23, 2020

O. Finkelman
Qwilt
S. Mishra
Verizon

November 20, 2019

CDNI Request Routing Extensions
draft-ietf-cdni-request-routing-extensions-08

Abstract

Open Caching architecture is a use case of Content Delivery Networks Interconnection (CDNI) in which the commercial Content Delivery Network (CDN) is the upstream CDN (uCDN) and the ISP caching layer serves as the downstream CDN (dCDN). The extensions specified in this document to the CDNI Metadata Interface (MI) and the Footprint and Capabilities Interface (FCI) are derived from requirements raised by Open Caching but are also applicable to CDNI use cases in general.

Table of Contents

1. Introduction

The Streaming Video Alliance [SVA] is a global association that works to solve streaming video challenges in an effort to improve end-user experience and adoption. The Open Caching Working Group [OCWG] of the Streaming Video Alliance [SVA] is focused on the delegation of video delivery requests from commercial CDNs to a caching layer at the Internet Service Provider's (ISP) network. Open Caching architecture is a specific use case of CDNI where the commercial CDN is the upstream CDN (uCDN) and the ISP caching layer is the downstream CDN (dCDN). The Open Caching Request Routing Specification [OC-RR] defines the Request Routing process and the interfaces that are required for its provisioning. This document defines and registers CDNI metadata object [RFC8006] and CDNI

Footprint and Capabilities object [RFC8008] that are required for Open Caching Request Routing. For consistency with other CDNI documents this document follows the CDNI convention of uCDN (upstream CDN) and dCDN (downstream CDN) to represent the commercial CDN and ISP caching layer respectively.

This document also registers CDNI Payload Types [RFC7736] for the defined objects:

- Redirect Target Capability (for dCDN advertising redirect target address)
- Fallback Target Metadata (for uCDN configuring fallback target address)

1.1. Terminology

The following terms are used throughout this document:

- FQDN - Fully Qualified Domain Name
- CDN - Content Delivery Network

Additionally, this document reuses the terminology defined in [RFC6707], [RFC7336], [RFC8006], [RFC8007], and [RFC8008]. Specifically, we use the following CDNI acronyms:

- FCI - Footprint and Capability Interface (see [RFC8008])
- MI - Metadata Interface (see [RFC8006])
- uCDN, dCDN - Upstream CDN and Downstream CDN respectively (see [RFC7336])
- RT - Redirection Target. Endpoint for redirection from uCDN to dCDN.
- RR - Request Router. An element responsible for routing user requests, typically using HTTP redirect or DNS CNAME, depending on the use case.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Redirect Target Capability

Iterative request redirection is defined in Section 1.1 of [RFC7336] and elaborated by examples in Sections 3.2 and 3.4 of [RFC7336]. A Redirection Target (RT) is defined in Section 2 of [RFC7975] for Recursive Request Redirection as:

> "The endpoint to which the User Agent is redirected. In CDNI, a RT may point to a number of different components, some examples include a surrogate in the same CDN as the request router, a request router in a dCDN, or a surrogate in a dCDN".

In this document we adopt the same definition of the RT for the Iterative Request Redirect use case. This use case requires the provisioning of the RT address to be used by the uCDN in order to redirect to the dCDN. RT addresses can vary between different footprints, for example, between different regions, and they may also change over time, for example as a result of network problems. Given this variable and dynamic nature of the redirect target address, it may not be suitable to advertise it during bootstrap. A more dynamic and footprint oriented interface is required. Section 4.3 of [RFC7336] suggests that it could be one of the roles of the FCI [RFC8008]. Following this suggestion, we have therefore, chosen to use the CDNI Footprint and Capabilities interface for redirect target address advertisement.

Use cases

- Footprint: The dCDN may want to have a different target per footprint. Note that a dCDN may spread across multiple geographies. This makes it easier to route client requests to a nearby request router. Though this can be achieved using a single canonical name and "Geo DNS", such that in different geographies the same hostname is resolved to different IP address, that approach has limitations; for example a client may be using a third party DNS resolver, making it impossible for the redirector to detect where the client is located, or Geo DNS granularity may be too rough for the requirement of the application.

- Scaling: The dCDN may choose to scale its request routing service by deploying more request routers in new locations and advertise them via an updatable interface like the FCI.

The Redirect Target capability object is used to indicate the target address the uCDN should use in order to redirect a client to the dCDN. A target may be attached to a specific uCDN host, a list of uCDN hosts, or used globally for all the hosts of the uCDN.

When a dCDN is attaching the redirect target to a specific uCDN host or a list of uCDN hosts, the dCDN MUST advertise the hosts within the Redirect Target capability object as "redirecting-hosts". In this case, the uCDN can redirect to that dCDN address, only if the User Agent request was to one of these uCDN hosts.

If the redirect target capability object does not contain a target or the target is empty, the uCDN MUST interpret it as "no target available for these uCDN hosts for the specified footprint". In case such a target was already advertised in a previous FCI object, the uCDN MUST interpret it as an update that deletes the previous redirect target.

2.1. DNS Redirect Target

A redirect target for DNS redirection is a FQDN used as an alias in a CNAME record response (see [RFC1034]) of the uCDN DNS router. Note that DNS routers make routing decisions based on either the DNS resolver's IP address or the client IP subnet when EDNS0 client-subnet (ECS) is used (see [RFC7871]). The dCDN may choose to advertise redirect targets and footprints to cover both cases, such that the uCDN resolution would route the DNS query to a different dCDN CNAMEs according client subnet or dCDN resolver IP address. This method further allows the dCDN DNS to optimize the resolution by localizing the target CNAMEs. A uCDN implementation SHOULD prefer routing based on client IP subnet when ECS option is present. A dCDN implementation using the ECS option MUST be aware of the privacy drawbacks listed in Section 2 of [RFC7871] and SHOULD follow the guidelines provided in Section 11.1 of [RFC7871].

2.2. HTTP Redirect Target

A redirect target for HTTP redirection is the URI to be used as the value for the Location header of a HTTP redirect 3xx response, typically a 302 (Found) (see Section 7.1.2 of [RFC7231] and section 6.4 of [RFC7231]).

2.3. Properties of Redirect Target Capability Object

The Redirect Target capability object consists of the following properties:

Property: redirecting-hosts

Description: One or more uCDN hosts to which this redirect target is attached. A redirecting host SHOULD be a host that was published in a HostMatch object by the uCDN as defined in Section 4.1.2 of [RFC8006].

Type: A list of Endpoint objects (see Section 4.3.3 of [RFC8006])

Mandatory-to-Specify: No. If not present, or empty, the redirect target applies to all hosts of the redirecting uCDN.

Property: dns-target

Description: Target CNAME record for DNS redirection.

Type: DnsTarget object (see Section 2.4)

Mandatory-to-Specify: No. If the dns-target is not present or empty the uCDN MUST interpret it as "no dns-target available".

Property: http-target

Description: Target URI for a HTTP redirect.

Type: HttpTarget object (see Section 2.5)

Mandatory-to-Specify: No. If the http-target is not present or empty the uCDN MUST interpret it as "no http-target available".

The following is an example of a Redirect Target capability object serialization that advertises a dCDN target address that is attached to a specific list of uCDN "redirecting-hosts". A uCDN host that is included in that list can redirect to the advertised dCDN redirect target. The capabilities object is serialized as a JSON object as defined in Section 5.1 of [RFC8008]

```
{
  "capabilities": [
    {
      "capability-type": "FCI.RedirectTarget",
      "capability-value": {
          "redirecting-hosts": [
             "a.service123.ucdn.example.com",
             "b.service123.ucdn.example.com"
          ],
          "dns-target": {
             "host": "service123.ucdn.dcdn.example.com"
          },
          "http-target": {
              "host": "us-east1.dcdn.example.com",
              "path-prefix": "/cache/1/",
              "include-redirecting-host": true
          }
      },
      "footprints": [
          <Footprint objects>
      ]
    }
  ]
}
```

2.4. DnsTarget Object

The DnsTarget object gives the target address for the DNS response to delegate from the uCDN to the dCDN.

Property: host

Description: The host property is a hostname or an IP address, without a port number.

Type: Endpoint object as defined in Section 4.3.3 of [RFC8006] with the limitation that it SHOULD NOT include a port number and, in case a port number is present, the uCDN MUST ignore it.

Mandatory-to-Specify: Yes.

2.4.1. DNS Target Example

The following is an example of DnsTarget object:

```
{
   "host": "service123.ucdn.dcdn.example.com"
}
```

The following is an example of a DNS query for uCDN address "a.service123.ucdn.example.com" and the corresponding CNAME redirection response:

```
Query:
a.service123.ucdn.example.com:
type A, class IN

Response:
NAME: a.service123.ucdn.example.com, TYPE: CNAME, CLASS: IN,
TTL: 120, RDATA: service123.ucdn.dcdn.example.com
```

2.5. HttpTarget Object

The HttpTarget object gives the necessary information to construct the target Location URI for HTTP redirection.

Property: host

Description: Hostname or IP address and an optional port, i.e., the host and port of the authority component of the URI as described in Section 3.2 of [RFC3986].

Type: Endpoint object as defined in Section 4.3.3 of [RFC8006].

Mandatory-to-Specify: Yes.

Property: scheme

Description: A URI scheme to be used in the redirect response location construction. When present, the uCDN MUST use the provided scheme in for HTTP redirection to the dCDN.

Type: A URI scheme as defined in Section 3.1 of [RFC3986] represented as a JSON string. The scheme MUST be either "http" or "https".

Mandatory-to-Specify: No. If this property is absent or empty the uCDN request router MUST use the same scheme as was used in the original request before redirection.

Property: path-prefix

Description: A path prefix for the HTTP redirect Location header. The original path is appended after this prefix.

Type: A prefix of a path-absolute as defined in Section 3.3 of [RFC3986]. The prefix MUST end with a trailing slash, to indicate the end of the last path segment in the prefix.

Mandatory-to-Specify: No. If this property is absent or empty, the uCDN MUST NOT prepend a path prefix to the original content path, i.e., the original path MUST appear in the location URI right after the authority component.

Property: include-redirecting-host

Description: A flag indicating whether or not to include the redirecting host as the first path segment after the path-prefix. If set to true and a "path-prefix" is used, the uCDN redirecting host MUST be added as a separate path segment after the path-prefix and before the original URL path. If set to true and there is no path-prefix, the uCDN redirecting host MUST be prepended as the first path segment in the redirect URL.

Type: Boolean.

Mandatory-to-Specify: No. Default value is False.

2.5.1. HTTP Target Example

Example of HttpTarget object with a "scheme", a "path-prefix", and "include-redirecting-host" properties:

```
{
   "host": "us-east1.dcdn.example.com",
   "scheme": "https",
   "path-prefix": "/cache/1/",
   "include-redirecting-host": true
}
```

Example of a HTTP request for content at uCDN host "a.service123.ucdn.example.com" and the corresponding HTTP response with a Location header, used for redirecting the client to the dCDN, constructed according to the HttpTarget object from the above example:

```
Request:
GET /vod/1/movie.mp4 HTTP/1.1
Host: a.service123.ucdn.example.com

Response:
HTTP/1.1 302 Found
Location: https://us-east1.dcdn.example.com/cache/1/
a.service123.ucdn.example.com/vod/1/movie.mp4
```

2.6. Usage Example

Before requests can be routed from the uCDN to the dCDN the CDNs must exchange service configurations between them. Using the MI, the uCDN advertises out-of-band its hosts to the dCDN, each host is designated by a hostname and has its own specific metadata (see Section 4.1.2 of [RFC8006]. The dCDN, using the FCI, advertises, also out-of-band, the redirect target address object defined in Section 2.3 for the relevant uCDN hosts. The following is a generalized example of the message flow between an upstream CDN and a downstream dCDN. For simplicity, we focus on the sequence of messages between the uCDN and dCDN and not on how they are passed.

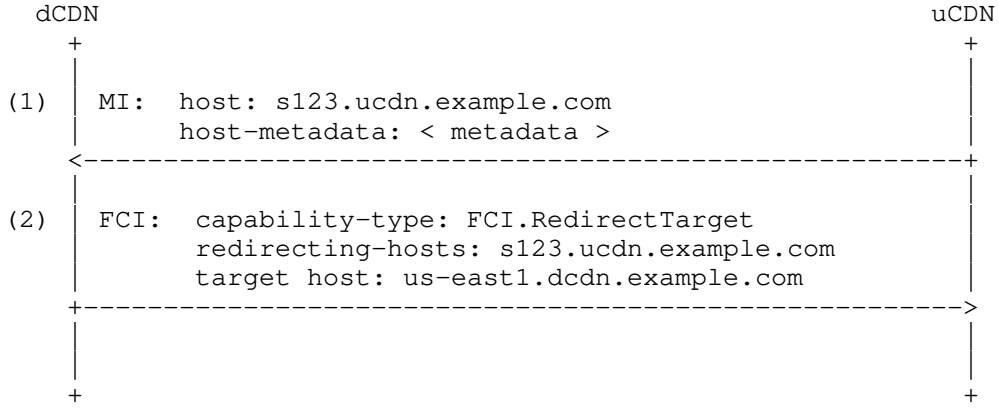


Figure 1: Redirect target address advertisement

1. The uCDN advertises a host (s123.ucdn.example.com) with the host metadata.

2. The dCDN advertises its FCI objects to the uCDN including a FCI.RedirectTarget object that contains the redirect target address (us-east1.dcdn.example.com) specified for that uCDN host.

Once the redirect target has been set, the uCDN can start redirecting user requests to the dCDN. The following is a generic sequence of

redirection using the host and redirect target that were advertised in Figure 1 above.

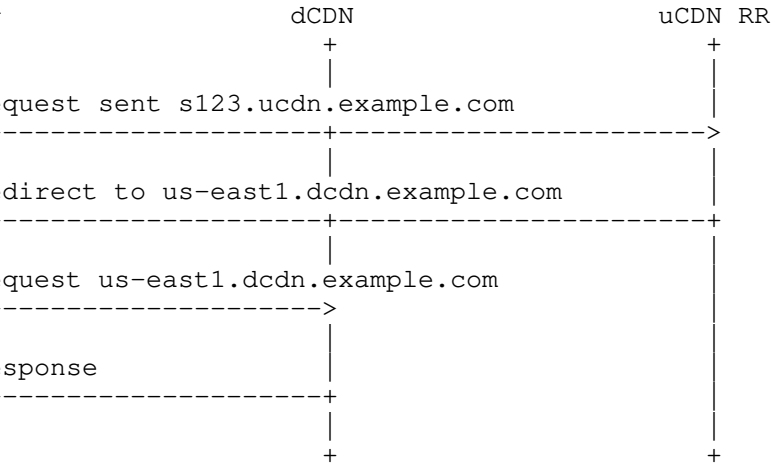


Figure 2: Generic requests redirection sequence

1. The End User sends a request (DNS or HTTP) to the uCDN Request Router (RR).

2. Using the previously advertised Redirect Target, the uCDN redirects the request to the dCDN.

3. The End User sends a request to the dCDN.

4. The dCDN either sends a response or reroutes it, for example, to a dCDN surrogate.

3. Fallback Target Address Metadata

Open Caching requires that the uCDN provides a fallback target server to the dCDN, to be used in cases where the dCDN cannot properly handle the request. To avoid redirect loops, the fallback target server's address at the uCDN MUST be different from the original uCDN address from which the client was redirected to the dCDN. The uCDN MUST avoid further redirection when receiving the client request at the fallback target. The fallback target is defined as a generic metadata object (see Section 3.2 of [RFC8006])

Use cases

- Failover: A dCDN request router receives a request but has no caches to which it can route the request. This can happen in the case of failures or temporary network overload.

- No coverage: A dCDN request router receives a request from a client located in an area inside the footprint but not covered by the dCDN caches or outside the dCDN footprint coverage. In such cases, the router may choose to redirect the request back to the uCDN fallback address.

- Error: A cache may receive a request that it cannot properly serve, for example, some of the metadata objects for that service were not properly acquired. In this case, the cache's "default action" may be to "redirect back to uCDN".

The Fallback target metadata object is used to indicate the target address the dCDN should redirect a client to when falling back to the uCDN. Fallback target address is represented as an endpoint object as defined in Section 4.3.3 of [RFC8006].

In DNS redirection a CNAME record is used as the fallback target address.

In HTTP redirection a hostname is used as the fallback target address.

When using HTTP redirect to route a client request back to the uCDN, it is the dCDN's responsibility to use the original URL path as the client would have used for the original uCDN request, stripping, if needed, the dCDN path-prefix and/or the uCDN hostname from the redirect URL that may have been used to request the content from the dCDN.

3.1. Properties of Fallback Target Address Metadata Object

The MI.FallbackTarget Metadata object consists of the following single property:

Property: host

Description: Target address to which the dCDN can redirect the client.

Type: Endpoint object as defined in Section 4.3.3 of [RFC8006] with the limitation that in case of DNS delegation it SHOULD NOT include a port number and, in case a port number is present, the dCDN MUST ignore it.

Mandatory-to-Specify: Yes.

Property: scheme

Description: A URI scheme to be used in the redirect response location construction. When present, the dCDN MUST use this scheme in case of HTTP redirection to the uCDN fallback address.

Type: A URI scheme as defined in Section 3.1 of [RFC3986] represented as a JSON string. The scheme MUST be either "http" or "https".

Mandatory-to-Specify: No. In case of HTTP redirection to fallback, if this property is absent or empty, the dCDN redirecting entity MUST use the same scheme as in the request received by the dCDN.

Example of a MI.FallbackTarget Metadata object that designates the host address the dCDN should use as fallback address to redirect back to the uCDN.

```
{
    "generic-metadata-type": "MI.FallbackTarget",
    "generic-metadata-value":
    {
        "host": "fallback-a.service123.ucdn.example",
        "scheme": "https"
    }
}
```

3.2. Usage Example

The uCDN advertises out-of-band the fallback target address to the dCDN, so that the dCDN may redirect a request back to the uCDN in case the dCDN cannot serve it. Using the MI the uCDN advertises its hosts to the dCDN, along with their specific host metadata (see Section 4.1.2 of [RFC8006]. The Fallback Target generic metadata object is encapsulated within the "host-metadata" property of each host. The following is an example of a message flow between an upstream CDN and a downstream dCDN. For simplicity, we focus on the sequence of messages between the uCDN and dCDN, not on how they are passed.

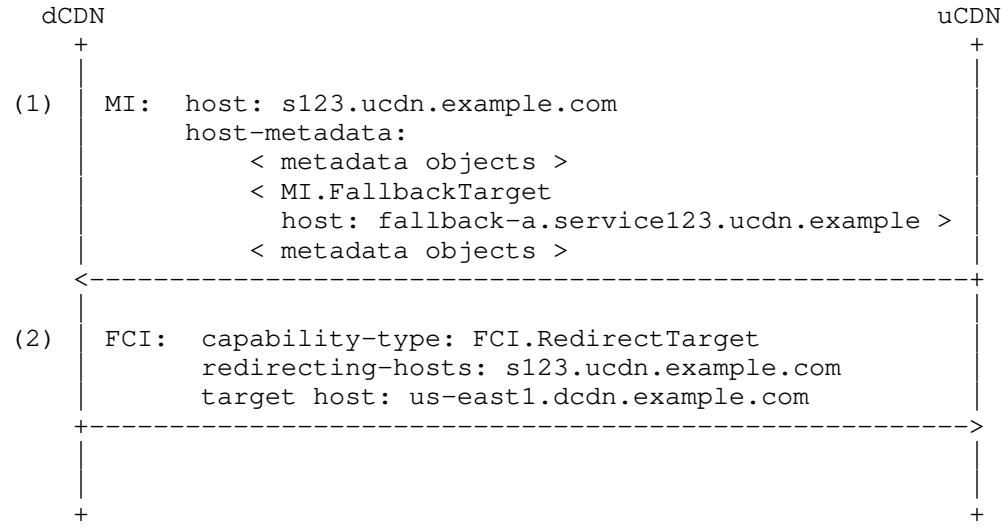


Figure 3: Advertisement of host metadata with Fallback Target

1. The uCDN advertises a host (s123.ucdn.example.com) with the host metadata. The host-metadata property contains a MI.FallbackTarget object.

2. The dCDN advertises its FCI objects to the uCDN including a FCI.RedirectTarget object that contains the redirect target address (us-east1.dcdn.example.com) specified for that uCDN host.

The following is a generic sequence of redirection using the configurations that were advertised in Figure 3 above. In this case the dCDN redirects back to the uCDN fallback target address.

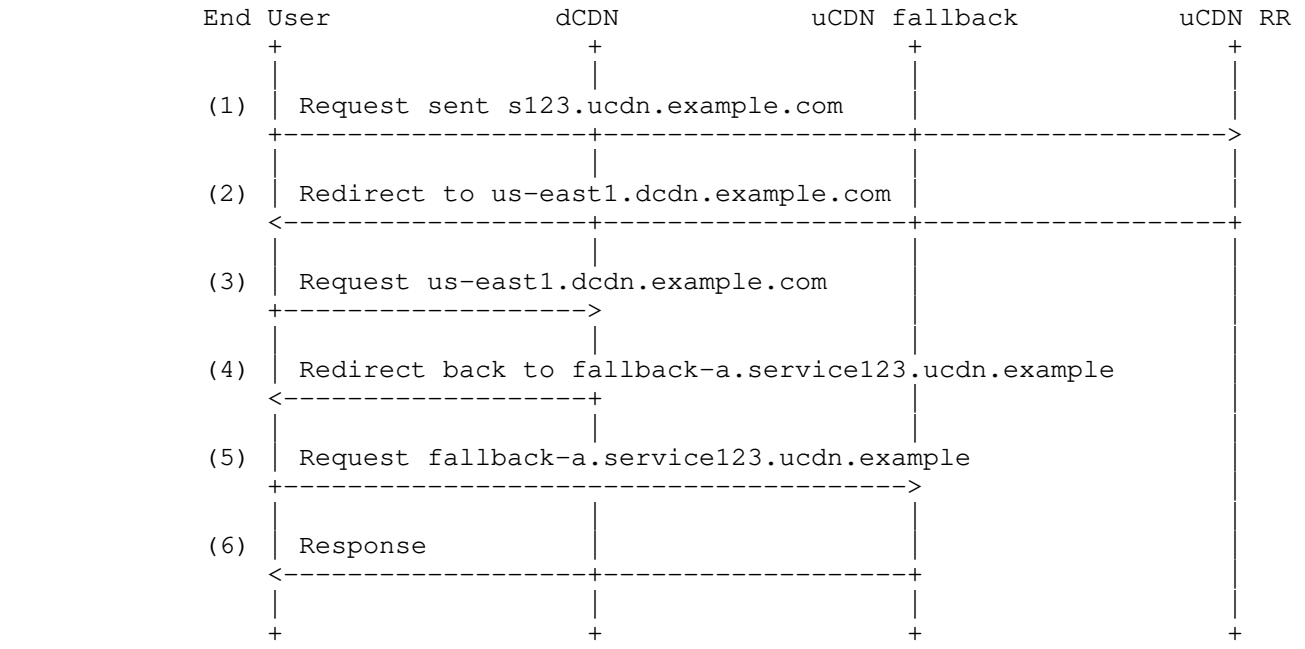


Figure 4: Redirection to Fallback Target

1. The End User sends a request (DNS or HTTP) to the uCDN Request Router (RR).

2. Using the previously advertised Redirect Target, the uCDN redirects the request to the dCDN.

3. The End User sends a request to the dCDN.

4. The dCDN cannot handled the request and, therefore, redirects it back to the uCDN fallback target address.

5. The End User sends the request to the uCDN fallback target address.

6. The uCDN either sends a response or reroutes it, for example, to a uCDN surrogate.

3.3. uCDN addressing considerations

When advertising fallback addresses to the dCDN the uCDN SHOULD consider the failure use cases that may lead the dCDN to route requests to uCDN fallback. In extreme dCDN network failures or under denial-of-service (DoS) attacks, requests coming from a large segment

or multiple segments of the dCDN may be routed back to the uCDN. The uCDN SHOULD therefore design its fallback addressing scheme and its available resources accordingly. A favorable approach would be for the uCDN to use different fallback target address for each uCDN host, enabling it to load balance the requests using the same methods as it would for its original hosts. See Sections 4.1.2 and 4.1.3 of [RFC8006] for a detailed description of how to use GenericMetadata objects within the HostMatch object advertised in the HostIndex of the uCDN.

4. IANA Considerations

4.1. CDNI Payload Types

This document requests the registration of the following CDNI Payload Types under the IANA "CDNI Payload Types" registry defined in [RFC7736]:

Payload Type	Specification
FCI.RedirectTarget	RFCthis
MI.FallbackTarget	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

4.1.1. CDNI FCI RedirectTarget Payload Type

Purpose: The purpose of this payload type is to distinguish RedirectTarget FCI objects

Interface: FCI

Encoding: see Section 2.3

4.1.2. CDNI MI FallbackTarget Payload Type

Purpose: The purpose of this payload type is to distinguish FallbackTarget MI objects (and any associated capability advertisement)

Interface: MI/FCI

Encoding: see Section 3.1

5. Security Considerations

This specification is in accordance with the CDNI Metadata Interface and the CDNI Request Routing: Footprint and Capabilities Semantics. As such, it is subject to the security and privacy considerations as defined in Section 8 of [RFC8006] and in Section 7 of [RFC8008] respectively.

5.1. Confidentiality and Privacy

The Redirect Target FCI object potentially reveals information about the internal structure of the dCDN network. A third party could intercept the FCI transactions and use the information to attack the dCDN. The same is also true for the Fallback Target Metadata object as it may reveal information about the internal structure of the uCDN, exposing it to external exploits. Implementations of the FCI and MI MUST therefore use strong authentication and encryption and strictly follow the directions for securing the interface as defined for the Metadata Interface in Section 8.3 of [RFC8006].

6. Acknowledgements

The authors thank Nir B. Sopher for reality checks against production use cases, his contribution is significant to this document. The authors also thank Ben Niven-Jenkins for his review and feedback and Kevin J. Ma for his guidance throughout the development of this document including his regular reviews.

7. References

7.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <https://www.rfc-editor.org/info/rfc7975>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8007] Murray, R. and B. Niven-Jenkins, "Content Delivery Network Interconnection (CDNI) Control Interface / Triggers", RFC 8007, DOI 10.17487/RFC8007, December 2016, <https://www.rfc-editor.org/info/rfc8007>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[OC-RR] Finkelman, O., Ed., Hofmann, J., Klein, E., Mishra, S., Ma, K., Sahar, D., and B. Zurat, "Open Caching - Request Routing Functional Specification", Version 1.1, October 2019, <https://www.streamingvideoalliance.org/books/open-cache-request-routing-functional-specification/>.

[OCWG] "Open Caching Home Page", <https://www.streamingvideoalliance.org/technical-groups/open-caching/>.

[RFC7736] Ma, K., "Content Delivery Network Interconnection (CDNI) Media Type Registration", RFC 7736, DOI 10.17487/RFC7736, December 2015, <https://www.rfc-editor.org/info/rfc7736>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/info/rfc7871>.

[SVA] "Streaming Video Alliance Home Page", <https://www.streamingvideoalliance.org>.

Authors' Addresses

Ori Finkelman
Qwilt
6, Ha'harash
Hod HaSharon 4524079
Israel

Email: ori.finkelman.ietf@gmail.com

Sanjay Mishra
Verizon
13100 Columbia Pike
Silver Spring, MD 20904
USA

Email: sanjay.mishra@verizon.com

CDNI
Internet-Draft
Intended status: Standards Track
Expires: April 10, 2020

R. van Brandenburg
Tiledmedia
K. Leung
Cisco Systems, Inc.
P. Sorber
Apple, Inc.

October 8, 2019

URI Signing for CDN Interconnection (CDNI)
draft-ietf-cdni-uri-signing-19

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a URI signing method as a JSON Web Token (JWT) profile.

The proposed URI signing method specifies the information needed to be included in the URI to transmit the signed JWT, as well as the claims needed by the signed JWT to authorize a UA. The mechanism described can be used both in CDNI and single CDN scenarios.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 10, 2020.

Table of Contents

1. Introduction

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of redirection between interconnected CDNs (CDNI) and between a Content Service Provider (CSP) and a CDN. The primary goal of URI Signing is to make sure that only authorized User Agents (UAs) are able to access the content, with a CSP being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as Digital Rights Management (DRM), are more appropriate. In addition to access control, URI Signing also has benefits in reducing the impact of denial-of-service attacks.

The overall problem space for CDN Interconnection (CDNI) is described in CDNI Problem Statement [RFC6707]. This document, along with the CDNI Requirements [RFC7337] document and the CDNI Framework [RFC7336], describes the need for interconnected CDNs to be able to implement an access control mechanism that enforces a CSP's distribution policies.

Specifically, the CDNI Framework [RFC7336] states:

> The CSP may also trust the CDN operator to perform actions such as delegating traffic to additional downstream CDNs, and to enforce per-request authorization performed by the CSP using techniques such as URI signing.

In particular, the following requirement is listed in the CDNI Requirements [RFC7337]:

> MI-16 {HIGH} The CDNI Metadata interface shall allow signaling of authorization checks and verification that are to be performed by the Surrogate before delivery. For example, this could potentially include the need to verify information (e.g., Expiry time, Client IP address) required for access authorization.

This document defines a method of signing URIs that allows Surrogates in interconnected CDNs to enforce a per-request authorization initiated by the CSP. Splitting the role of initiating per-request authorization by the CSP and the role of verifying this authorization by the CDN allows any arbitrary distribution policy to be enforced across CDNs without the need of CDNs to have any awareness of the specific CSP distribution policies.

The method is implemented using Signed JSON Web Tokens (JWTs) [RFC7519].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the terminology defined in the CDNI Problem Statement [RFC6707].

This document also uses the terminology of the JSON Web Token (JWT) [RFC7519].

In addition, the following terms are used throughout this document:

- Signed URI: A URI for which a signed JWT is provided.
- Target CDN URI: URI created by the CSP to direct a UA towards the Upstream CDN (uCDN). The Target CDN URI can be signed by the CSP and verified by the uCDN and possibly further Downstream CDNs (dCDNs).

- Redirection URI: URI created by the uCDN to redirect a UA towards the dCDN. The Redirection URI can be signed by the uCDN and verified by the dCDN. In a cascaded CDNI scenario, there can be more than one Redirection URI.

- Signed Token Renewal: A series of signed JWTs that are used for subsequent access to a set of related resources in a CDN, such as a set of HTTP Adaptive Streaming files. Every time a signed JWT is used to access a particular resource, a new signed JWT is sent along with the resource that can be used to request the next resource in the set. When generating a new signed JWT in Signed Token Renewal, parameters are carried over from one signed JWT to the next.

1.2. Background and overview on URI Signing

A CSP and CDN are assumed to have a trust relationship that enables the CSP to authorize access to a content item by including a set of claims in the form of a signed JWT in the URI before redirecting a UA to the CDN. Using these attributes, it is possible for a CDN to check an incoming content request to see whether it was authorized by the CSP (e.g., based on the UA's IP address or a time window). To prevent the UA from altering the claims a JWT MUST be signed.

Figure 1, shown below, presents an overview of the URI Signing mechanism in the case of a CSP with a single CDN. When the UA browses for content on CSP's website (#1), it receives HTML web pages with embedded content URIs. Upon requesting these URIs, the CSP redirects to a CDN, creating a Target CDN URI (#2) (alternatively, the Target CDN URI itself is embedded in the HTML). The Target CDN URI is the Signed URI which may include the IP address of the UA and/or a time window. The signed URI always contains a signed JWT generated by the CSP using a shared secret or private key. Once the UA receives the response with the Signed URI, it sends a new HTTP request using the Signed URI to the CDN (#3). Upon receiving the request, the CDN authenticates the Signed URI by verifying the signed JWT. If applicable, the CDN checks whether the source IP address of the HTTP request matches the one in the Signed URI and/or if the time window is still valid. After these claims are verified, the CDN delivers the content (#4).

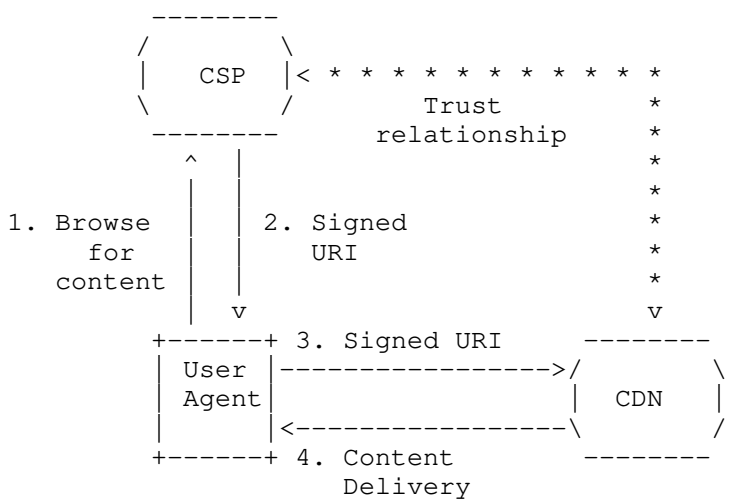


Figure 1: Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, as shown in Figure 2 below, URI Signing operates the same way in the initial steps #1 and #2 but the later steps involve multiple CDNs delivering the content. The main difference from the single CDN case is a redirection step between the uCDN and the dCDN. In step #3, the UA may send an HTTP request or a DNS request. Depending on whether HTTP-based or DNS-based request routing is used. The uCDN responds by directing the UA towards the dCDN using either a Redirection URI (i.e., a Signed URI generated by the uCDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the dCDN (#5). The received URI is verified by the dCDN before delivering the content (#6). Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 5).

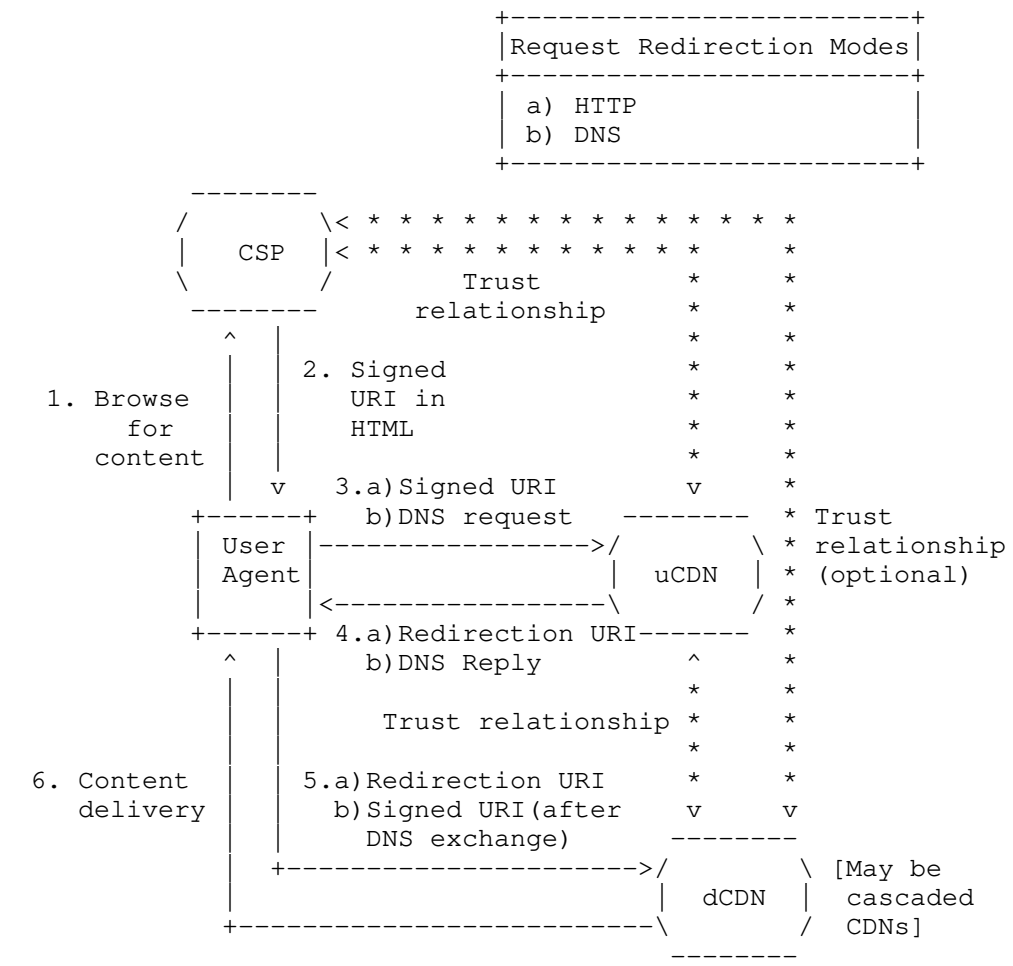


Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, uCDN, and dCDN have direct implications for URI Signing. In the case shown in Figure 2, the CSP has a trust relationship with the uCDN. The delivery of the content may be delegated to a dCDN, which has a relationship with the uCDN but may have no relationship with the CSP.

In CDNI, there are two methods for request routing: DNS-based and HTTP-based. For DNS-based request routing, the Signed URI (i.e., the Target CDN URI) provided by the CSP reaches the dCDN directly. In the case where the dCDN does not have a trust relationship with the CSP, this means that either an asymmetric public/private key method needs to be used for computing the signed JWT (because the CSP and

dCDN are not able to exchange symmetric shared secret keys), or the CSP needs to allow the uCDN to redistribute shared keys to a subset of their dCDNs.

For HTTP-based request routing, the Signed URI (i.e., the Target CDN URI) provided by the CSP reaches the uCDN. After this URI has been verified by the uCDN, the uCDN creates and signs a new Redirection URI, redirecting the UA to the dCDN. Since this new URI can have a new signed JWT, the relationship between the dCDN and CSP is not relevant. Because a relationship between uCDN and dCDN always exists, either asymmetric public/private keys or symmetric shared secret keys can be used for URI Signing with HTTP-based request routing. Note that the signed Redirection URI MUST maintain the same (or higher) level of security as the original Signed URI.

Mode	Asymmetric Key	Symmetric Key
HTTP	Public key (uCDN)	Shared key (uCDN)
DNS	Public key (CSP)	Shared key (CSP)

Figure 3: CDNI URI Signing Key

Two types of keys can be used for URI Signing: asymmetric keys and symmetric keys. Asymmetric keys are based on a public/private key pair mechanism and always contain a private key known only to the entity signing the URI (either CSP or uCDN) and a public key for the verification of the Signed URI. With symmetric keys, the same key is used by both the signing entity for signing the URI as well as by the verifying entity for verifying the Signed URI. Regardless of the type of keys used, the verifying entity has to obtain the key (either the public or the symmetric key). There are very different requirements (outside the scope of this document) for distributing asymmetric keys and symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent third parties from getting access to the key, since they could then generate valid Signed URIs for unauthorized requests. Key distribution for asymmetric keys does not require confidentiality since public keys can typically be distributed openly (because they cannot be used to sign URIs) and private keys are kept secret by the URI signer.

1.4. URI Signing in a non-CDNI context

While the URI signing method defined in this document was primarily created for the purpose of allowing URI Signing in CDNI scenarios, i.e., between a uCDN and a dCDN, there is nothing in the defined URI Signing method that precludes it from being used in a non-CDNI

context. As such, the described mechanism could be used in a single-CDN scenario such as shown in Figure 1 in Section 1.2, for example to allow a CSP that uses different CDNs to only have to implement a single URI Signing mechanism.

2. JWT Format and Processing Requirements

The concept behind URI Signing is based on embedding a signed JSON Web Token (JWT) [RFC7519] in an HTTP or HTTPS URI [RFC7230] (Section 2.7). The signed JWT contains a number of claims that can be verified to ensure the UA has legitimate access to the content.

This document specifies the following attribute for embedding a signed JWT in a Target CDN URI or Redirection URI:

- URI Signing Package (URISigningPackage): The URI attribute that encapsulates all the URI Signing claims in a signed JWT encoded format. This attribute is exposed in the Signed URI as a URI query parameter or as a URL path parameter.

The parameter name of the URI Signing Package Attribute is defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include a parameter name for the URI Signing Package Attribute, the parameter name can be set by configuration (out of scope of this document).

The URI Signing Package will be found by searching the URI, left-to-right, for the following sequence:

- a reserved character (as defined in [RFC3986] Section 2.2),
- the URI Signing Package Attribute name,
- if the last character of the URI Signing Package Attribute name is not a reserved character, an equal symbol ('='),
- and a sequence of zero or more non-reserved characters that will be interpreted as a signed JWT,
- terminated by either a reserved character or the end of the URI.

The first such match will be taken to provide the signed JWT; the URI will not be searched for multiple signed JWTs.

2.1. JWT Claims

This section identifies the set of claims that can be used to enforce the CSP distribution policy. New claims can be introduced in the future to extend the distribution policy capabilities.

In order to provide distribution policy flexibility, the exact subset of claims used in a given signed JWT is a runtime decision. Claim requirements are defined in the CDNI Metadata (Section 4.4). If the CDNI Metadata interface is not used, or does not include claim requirements, the claim requirements can be set by configuration (out of scope of this document).

The following claims (where the "JSON Web Token Claims" registry claim name is specified in parenthesis below) are used to enforce the distribution policies. All of the listed claims are mandatory to implement in a URI Signing implementation, but are not mandatory to use in a given signed JWT. (The "optional" and "mandatory" identifiers in square brackets refer to whether or not a given claim MUST be present in a URI Signing JWT.) A CDN MUST be able to parse and process all of the claims listed below.

Note: See the Security Considerations (Section 7) section on the limitations of using an expiration time and client IP address for distribution policy enforcement.

2.1.1. Issuer (iss) claim

Issuer (iss) [optional] - The semantics in [RFC7519] Section 4.1.1 MUST be followed. This claim MAY be used to verify authorization of the issuer of a signed JWT and also MAY be used to confirm that the indicated key was provided by said issuer. If the CDN verifying the signed JWT does not support Issuer verification, or if the Issuer in the signed JWT does not match the list of known acceptable Issuers, the CDN MUST reject the request. If the received signed JWT contains an Issuer claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Issuer claim, and the Issuer value MUST be updated to identify the redirecting CDN. If the received signed JWT does not contain an Issuer claim, an Issuer claim MAY be added to a signed JWT generated for CDNI redirection.

2.1.2. Subject (sub) claim

Subject (sub) [optional] - The semantics in [RFC7519] Section 4.1.2 MUST be followed. If this claim is used, it MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form, because it contains personally identifiable information. This claim contains information about the subject (for example, a user or an

agent) that MAY be used to verify the signed JWT. If the received signed JWT contains a Subject claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Subject claim, and the Subject value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Subject claim if no Subject claim existed in the received signed JWT.

2.1.3. Audience (aud) claim

Audience (aud) [optional] - The semantics in [RFC7519] Section 4.1.3 MUST be followed. This claim is used to ensure that the CDN verifing the JWT is an intended recipient of the request. The claim should contain an identity on behalf of whom the CDN can verify the token (e.g., the CSP or any uCDN in the chain). A dCDN MAY modify the claim as long it can generate a valid signature.

2.1.4. Expiry Time (exp) claim

Expiry Time (exp) [optional] - The semantics in [RFC7519] Section 4.1.4 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the CDN verifying the signed JWT does not support Expiry Time verification, or if the Expiry Time in the signed JWT corresponds to a time equal to or earlier than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains a Expiry Time claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Expiry Time claim, and the Expiry Time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add an Expiry Time claim if no Expiry Time claim existed in the received signed JWT.

2.1.5. Not Before (nbf) claim

Not Before (nbf) [optional] - The semantics in [RFC7519] Section 4.1.5 MUST be followed, though URI Signing implementations MUST NOT allow for any time synchronization "leeway". Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that the CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the CDN verifying the signed JWT does not support Not Before time verification, or if the Not Before time in the signed JWT corresponds to a time later than the time of the content request, the CDN MUST reject the request. If the received signed JWT contains a Not Before time claim, then any JWT

subsequently generated for CDNI redirection MUST also contain a Not Before time claim, and the Not Before time value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Not Before time claim if no Not Before time claim existed in the received signed JWT.

2.1.6. Issued At (iat) claim

Issued At (iat) [optional] - The semantics in [RFC7519] Section 4.1.6 MUST be followed. Note: The time on the entities that generate and verify the signed URI SHOULD be in sync. In the CDNI case, this means that CSP, uCDN, and dCDN servers need to be time-synchronized. It is RECOMMENDED to use NTP [RFC5905] for time synchronization. If the received signed JWT contains an Issued At claim, then any JWT subsequently generated for CDNI redirection MUST also contain an Issued At claim, and the Issuer value MUST be updated to identify the time the new JWT was generated. If the received signed JWT does not contain an Issued At claim, an Issued At claim MAY be added to a signed JWT generated for CDNI redirection.

2.1.7. Nonce (jti) claim

Nonce (jti) [optional] - The semantics in [RFC7519] Section 4.1.7 MUST be followed. A Nonce can be used to prevent replay attacks if the CDN stores a list of all previously used Nonce values, and verifies that the Nonce in the current JWT has never been used before. If the signed JWT contains a Nonce claim and the CDN verifying the signed JWT either does not support Nonce storage or has previously seen the nonce used in a request for the same content, then the CDN MUST reject the request. If the received signed JWT contains a Nonce claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Nonce claim, and the Nonce value MUST be the same as in the received signed JWT. If the received signed JWT does not contain a Nonce claim, a Nonce claim MUST NOT be added to a signed JWT generated for CDNI redirection.

2.1.8. CDNI Claim Set Version (cdniv) claim

CDNI Claim Set Version (cdniv) [optional] - The CDNI Claim Set Version (cdniv) claim provides a means within a signed JWT to tie the claim set to a specific version of this specification. The cdniv claim is intended to allow changes in and facilitate upgrades across specifications. The type is JSON integer and the value MUST be set to "1", for this version of the specification. In the absence of this claim, the value is assumed to be "1". For future versions this claim will be mandatory. Implementations MUST reject signed JWTs with unsupported CDNI Claim Set versions.

2.1.9. CDNI Critical Claims Set (cdnicrit) claim

CDNI Critical Claims Set (cdnicrit) [optional] - The CDNI Critical Claims Set (cdnicrit) claim indicates that extensions to this specification are being used that MUST be understood and processed. Its value is a comma separated listing of claims in the Signed JWT that use those extensions. If any of the listed extension claims are not understood and supported by the recipient, then the Signed JWT is invalid. Producers MUST NOT include claim names defined by this specification, duplicate names, or names that do not occur as claim names within the Signed JWT in the cdnicrit list. Producers MUST NOT use the empty list "" as the cdnicrit value. Recipients MAY consider the Signed JWT to be invalid if the cdnicrit list contains any claim names defined by this specification or if any other constraints on its use are violated. This claim MUST be understood and processed by all implementations.

2.1.10. Client IP (cdniip) claim

Client IP (cdniip) [optional] - The Client IP (cdniip) claim hold an IP address or IP prefix for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 addresses [RFC0791] or canonical text representation for IPv6 addresses [RFC5952]. The request MUST be rejected if sourced from a client outside of the specified IP range. Since the client IP is considered personally identifiable information this field MUST be a JSON Web Encryption (JWE [RFC7516]) Object in compact serialization form. If the CDN verifying the signed JWT does not support Client IP verification, or if the Client IP in the signed JWT does not match the source IP address in the content request, the CDN MUST reject the request. The type of this claim is a JSON string that contains the JWE. If the received signed JWT contains a Client IP claim, then any JWT subsequently generated for CDNI redirection MUST also contain a Client IP claim, and the Client IP value MUST be the same as in the received signed JWT. A signed JWT generated for CDNI redirection MUST NOT add a Client IP claim if no Client IP claim existed in the received signed JWT.

2.1.11. CDNI URI Container (cdniuc) claim

URI Container (cdniuc) [optional] - The URI Container (cdniuc) holds the URI representation before a URI Signing Package is added. This representation can take one of several forms detailed in Section 2.1.15. If the URI Container used in the signed JWT does not match the URI of the content request, the CDN verifying the signed JWT MUST reject the request. When comparing the URI, the percent encoded form as defined in [RFC3986] Section 2.1 MUST be used. When redirecting a URI, the CDN generating the new signed JWT MAY change

the URI Container to comport with the URI being used in the redirection.

2.1.12. CDNI Expiration Time Setting (cdniets) claim

CDNI Expiration Time Setting (cdniets) [optional] - The CDNI Expiration Time Setting (cdniets) claim provides a means for setting the value of the Expiry Time (exp) claim when generating a subsequent signed JWT in Signed Token Renewal. Its type is a JSON numeric value. It denotes the number of seconds to be added to the time at which the JWT is verified that gives the value of the Expiry Time (exp) claim of the next signed JWT. The CDNI Expiration Time Setting (cdniets) SHOULD NOT be used when not using Signed Token Renewal and MUST be present when using Signed Token Renewal.

2.1.13. CDNI Signed Token Transport (cdnistt) claim

CDNI Signed Token Transport (cdnistt) [optional] - The CDNI Signed Token Transport (cdnistt) claim provides a means of signalling the method through which a new signed JWT is transported from the CDN to the UA and vice versa for the purpose of Signed Token Renewal. Its type is a JSON integer. Values for this claim are defined in Section 6.5. If using this claim you MUST also specify a CDNI Expiration Time Setting (cdniets) as noted above.

2.1.14. CDNI Signed Token Depth (cdnistd) claim

CDNI Signed Token Depth (cdnistd) [optional] - The CDNI Signed Token Depth (cdnistd) claim is used to associate a subsequent signed JWT, generated as the result of a CDNI Signed Token Transport claim, with a specific URI subset. Its type is a JSON integer. Signed JWTs MUST NOT use a negative value for the CDNI Signed Token Depth claim.

If the transport used for Signed Token Transport allows the CDN to associate the path component of a URI with tokens (e.g., an HTTP Cookie Path as described in section 4.1.2.4 of [RFC6265]), the CDNI Signed Token Depth value is the number of path segments that should be considered significant for this association. A CDNI Signed Token Depth of zero means that the client SHOULD be directed to return the token with requests for any path. If the CDNI Signed Token Depth is greater than zero, then the client SHOULD be directed to return the token for future requests wherein the first CDNI Signed Token Depth segments of the path match the first CDNI Signed Token Depth segments of the signed URI path. This matching MUST use the URI with the token removed, as specified in Section 2.1.15.

If the URI path to match contains fewer segments than the CDNI Signed Token Depth claim, a signed JWT MUST NOT be generated for the

purposes of Signed Token Renewal. If the CDNI Signed Token Depth claim is omitted, it means the same thing as if its value were zero. If the received signed JWT contains a CDNI Signed Token Depth claim, then any JWT subsequently generated for CDNI redirection or Signed Token Transport MUST also contain a CDNI Signed Token Depth claim, and the value MUST be the same as in the received signed JWT.

2.1.15. URI Container Forms

The URI Container (cdniuc) claim takes one of the following forms: 'hash:' or 'regex:'. More forms may be added in the future to extend the capabilities.

Before comparing a URI with contents of this container, the following steps MUST be performed:

- Prior to verification, remove the signed JWT from the URI. This removal is only for the purpose of determining if the URI matches; all other purposes will use the original URI. If the signed JWT is terminated by anything other than a sub-delimiter (as definined in [RFC3986] Section 2.2), everything from the reserved character (as defined in [RFC3986] Section 2.2) that precedes the URI Signing Package Attribute to the last character of the signed JWT will be removed, inclusive. Otherwise, everything from the first character of the URI Signing Package Attribute to the sub-delimiter that terminates the signed JWT will be removed, inclusive.

- Normalize the URI according to section 2.7.3 [RFC7230] and sections 6.2.2 and 6.2.3 [RFC3986]. This applies to both generation and verification of the signed JWT.

2.1.15.1. URI Hash Container (hash:)

Prefixed with 'hash:', this string is a URL Segment form ([RFC6920] Section 5) of the URI.

2.1.15.2. URI Regular Expression Container (regex:)

Prefixed with 'regex:', this string is any POSIX Section 9 [POSIX.1] Extended Regular Expression compatible regular expression used to match against the requested URI. These regular expressions MUST be evaluated in the POSIX locale (POSIX Section 7.2 [POSIX.1]).

Note: Because '\' has special meaning in JSON [RFC8259] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

An example of a 'regex:' is the following:

```
[^:]*\\://[^/]*/folder/content/quality_[^/]*/segment.{3}\\.mp4(\\?.*)?
```

Note: Due to computational complexity of executing arbitrary regular expressions, it is RECOMMENDED to only execute after verifying the JWT to ensure its authenticity.

2.2. JWT Header

The header of the JWT MAY be passed via the CDNI Metadata interface instead of being included in the URISigningPackage. The header value must be transmitted in the serialized encoded form and prepended to the JWT payload and signature passed in the URISigningPackage prior to verification. This reduces the size of the signed JWT token.

3. URI Signing Token Renewal

3.1. Overview

For content that is delivered via HTTP in a segmented fashion, such as MPEG-DASH [MPEG-DASH] or HTTP Live Streaming (HLS) [RFC8216], special provisions need to be made in order to ensure URI Signing can be applied. In general, segmented protocols work by breaking large objects (e.g., videos) into a sequence of small independent segments. Such segments are then referenced by a separate manifest file, which either includes a list of URLs to the segments or specifies an algorithm through which a User Agent can construct the URLs to the segments. Requests for segments therefore originate from the manifest file and, unless the URLs in the manifest file point to the CSP, are not subjected to redirection and URI Signing. This opens up a vulnerability to malicious User Agents sharing the manifest file and deep-linking to the segments.

One method for dealing with this vulnerability would be to include, in the manifest itself, Signed URIs that point to the individual segments. There exist a number of issues with that approach. First, it requires the CDN delivering the manifest to rewrite the manifest file for each User Agent, which would require the CDN to be aware of the exact segmentation protocol used. Secondly, it could also require the expiration time of the Signed URIs to be valid for an extended duration if the content described by the manifest is meant to be consumed in real time. For instance, if the manifest file were to contain a segmented video stream of more than 30 minutes in length, Signed URIs would require to be valid for a at least 30 minutes, thereby reducing their effectiveness and that of the URI Signing mechanism in general. For a more detailed analysis of how

segmented protocols such as HTTP Adaptive Streaming protocols affect CDNI, see Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

The method described in this section allows CDNs to use URI Signing for segmented content without having to include the Signed URIs in the manifest files themselves.

3.2. Signed Token Renewal mechanism

In order to allow for effective access control of segmented content, the URI signing mechanism defined in this section is based on a method through which subsequent segment requests can be linked together. As part of the JWT verification procedure, the CDN can generate a new signed JWT that the UA can use to do a subsequent request. More specifically, whenever a UA successfully retrieves a segment, it receives, in the HTTP 2xx Successful message, a signed JWT that it can use whenever it requests the next segment. As long as each successive signed JWT is correctly verified before a new one is generated, the model is not broken and the User Agent can successfully retrieve additional segments. Given the fact that with segmented protocols, it is usually not possible to determine a priori which segment will be requested next (i.e., to allow for seeking within the content and for switching to a different representation), the Signed Token Renewal uses the URI Regular Expression Container scoping mechanisms in the URI Container (cdniuc) claim to allow a signed JWT to be valid for more than one URL.

In order for this renewal of signed JWTs to work, it is necessary for a UA to extract the signed JWT from the HTTP 2xx Successful message of an earlier request and use it to retrieve the next segment. The exact mechanism by which the client does this is outside the scope of this document. However, in order to also support legacy UAs that do not include any specific provisions for the handling of signed JWTs, the following section defines a mechanism using HTTP Cookies [RFC6265] that allows such UAs to support the concept of renewing signed JWTs without requiring any additional UA support.

3.2.1. Required Claims

The cdnistt claim (Section 2.1.13) and cdniets claim (Section 2.1.12) MUST both be present for Signed Token Renewal. You MAY set cdnistt to a value of '0' to mean no Signed Token Renewal, but you still MUST have a corresponding cdniets that verifies as a JSON number. However, if you do not want to use Signed Token Renewal, it is RECOMMENDED to simply omit both.

3.3. Communicating a signed JWTs in Signed Token Renewal

This section assumes the value of the CDNI Signed Token Transport (cdnistt) claim has been set to 1. Other values of cdnistt are out of scope of this document.

When using the Signed Token Renewal mechanism, the signed JWT is transported to the UA via a 'URISigningPackage' cookie added to the HTTP 2xx Successful message along with the content being returned to the UA, or to the HTTP 3xx Redirection message in case the UA is redirected to a different server.

3.3.1. Support for cross-domain redirection

For security purposes, the use of cross-domain cookies is not supported in some application environments. As a result, the Cookie-based method for transport of the Signed Token described in the previous section might break if used in combination with an HTTP 3xx Redirection response where the target URL is in a different domain. In such scenarios, Signed Token Renewal of a signed JWT SHOULD be communicated via the query string instead, in a similar fashion to how regular signed JWTs (outside of Signed Token Renewal) are communicated. Note that the use of URL embedded signed JWTs SHOULD NOT be used in HTTP 2xx Successful messages, since UAs might not know how to extract the signed JWTs.

Note that the process described herein only works in cases where both the manifest file and segments constituting the segmented content are delivered from the same domain. In other words, any redirection between different domains needs to be carried out while retrieving the manifest file.

4. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. A dCDN that supports URI Signing needs to be able to advertise this capability to the uCDN. The uCDN needs to select a dCDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the uCDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the dCDN to verify a Signed URI. Events that pertain to URI Signing (e.g., request denial or delivery after access authorization) need to be included in the logs communicated through the CDNI Logging interface.

4.1. CDNI Control Interface

URI Signing has no impact on this interface.

4.2. CDNI Footprint & Capabilities Advertisement Interface

The CDNI Request Routing: Footprint and Capabilities Semantics document [RFC8008] defines support for advertising CDNI Metadata capabilities, via CDNI Payload Type. The CDNI Payload Type registered in Section 6.1 can be used for capability advertisement.

4.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [RFC7975] describes the recursive request redirection method. For URI Signing, the uCDN signs the URI provided by the dCDN. URI Signing therefore has has no impact on this interface.

4.4. CDNI Metadata Interface

The CDNI Metadata Interface [RFC8006] describes the CDNI metadata distribution needed to enable content acquisition and delivery. For URI Signing, a new CDNI metadata object is specified.

The UriSigning Metadata object contains information to enable URI signing and verification by a dCDN. The UriSigning properties are defined below.

Property: enforce

Description: URI Signing enforcement flag. Specifically, this flag indicates if the access to content is subject to URI Signing. URI Signing requires the dCDN to ensure that the URI is signed and verified before delivering content. Otherwise, the dCDN does not perform verification, regardless of whether or not the URI is signed.

Type: Boolean

Mandatory-to-Specify: No. The default is true.

Property: issuers

Description: A list of valid Issuers against which the Issuer claim in the signed JWT may be verified.

Type: Array of Strings

Mandatory-to-Specify: No. The default is an empty list. An empty list means that any Issuer is acceptable.

Property: package-attribute

Description: The name to use for the URI Signing Package.

Type: String

Mandatory-to-Specify: No. The default is "URISigningPackage".

Property: jwt-header

Description: The header part of JWT that is used for generating or verifying a signed JWT when the JWT token in the URI Signing Package does not contain a header part.

Type: String

Mandatory-to-Specify: No. By default, the header is assumed to be included in the JWT token.

The following is an example of a URI Signing metadata payload with all default values:

```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value": {}
}
```

The following is an example of a URI Signing metadata payload with explicit values:

```
{
  "generic-metadata-type": "MI.UriSigning"
  "generic-metadata-value":
    {
      "enforce": true,
      "issuers": ["csp", "ucdn1", "ucdn2"],
      "package-attribute": "usp",
      "jwt-header":
        {
            "alg": "ES256",
            "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWkldueaImf0"
        }
    }
}
```

4.5. CDNI Logging Interface

For URI Signing, the dCDN reports that enforcement of the access control was applied to the request for content delivery. When the request is denied due to enforcement of URI Signing, the reason is logged.

The following CDNI Logging field for URI Signing SHOULD be supported in the HTTP Request Logging Record as specified in CDNI Logging Interface [RFC7937], using the new "cdni_http_request_v2" record-type registered in Section 6.2.1.

- s-uri-signing (mandatory):
  - format: 3DIGIT
  - field value: this characterises the URI signing verification performed by the Surrogate on the request. The allowed values are registered in Section 6.4.
  - occurrence: there MUST be zero or exactly one instance of this field.
- s-uri-signing-deny-reason (optional):
  - format: QSTRING
  - field value: a string for providing further information in case the signed JWT was rejected, e.g., for debugging purposes.
  - occurrence: there MUST be zero or exactly one instance of this field.

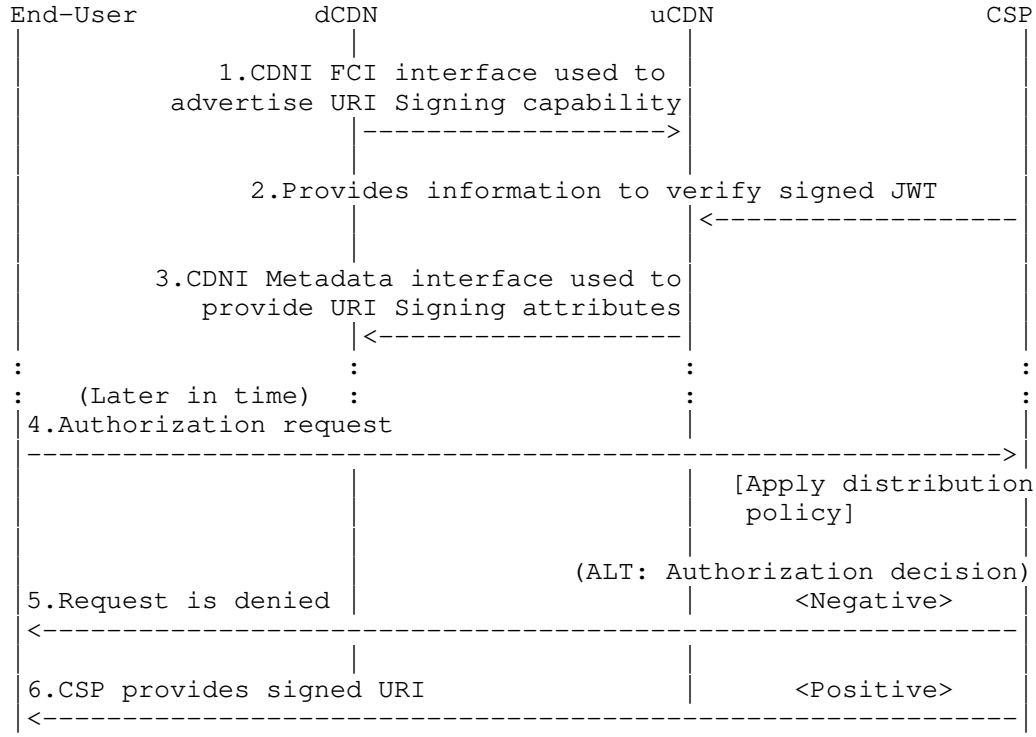
5. URI Signing Message Flow

URI Signing supports both HTTP-based and DNS-based request routing. JSON Web Token (JWT) [RFC7519] defines a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a signed JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

5.1. HTTP Redirection

For HTTP-based request routing, a set of information that is unique to a given end user content request is included in a signed JWT, using key information that is specific to a pair of adjacent CDNI hops (e.g., between the CSP and the uCDN or between the uCDN and a dCDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing method (assuming HTTP redirection, iterative request routing, and a CDN path with two CDNs) includes the following steps:



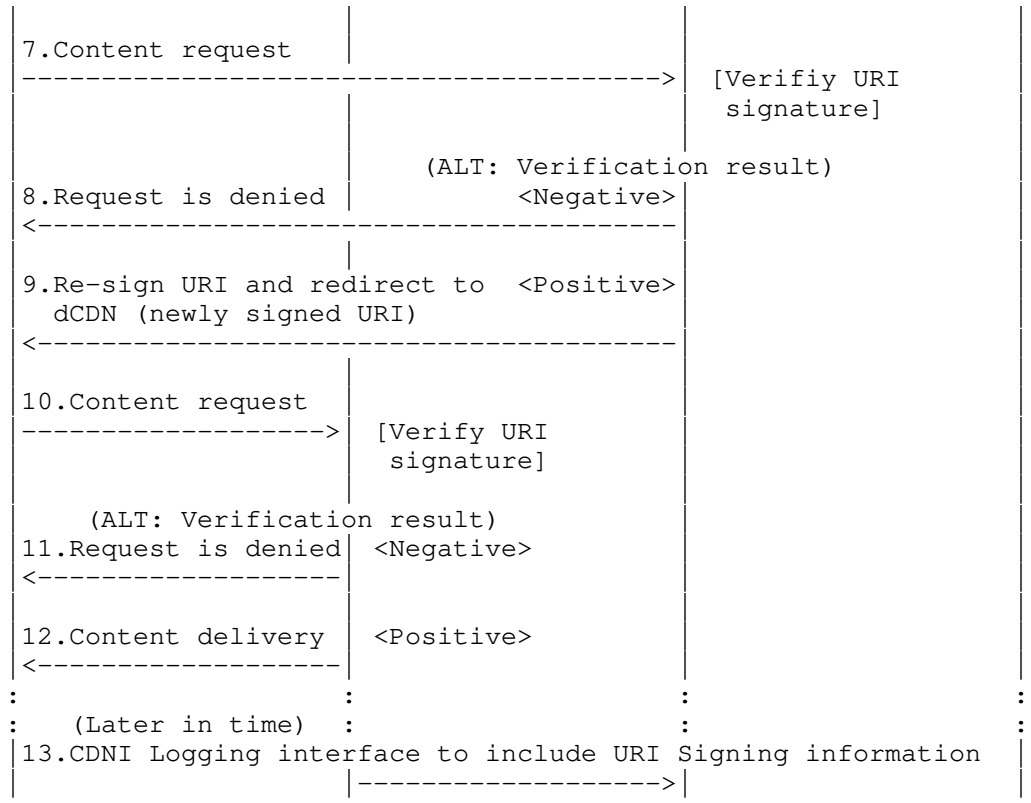


Figure 4: HTTP-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.

2. CSP provides to the uCDN the information needed to verify signed JWTs from that CSP. For example, this information may include a key value.

3. Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify signed JWTs from the uCDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute.

4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its personal distribution policy.

5. If the authorization decision is negative, the CSP rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.

6. If the authorization decision is positive, the CSP computes a Signed URI that is based on unique parameters of that request and conveys it to the end user as the URI to use to request the content.

7. On receipt of the corresponding content request, the uCDN verifies the signed JWT in the URI using the information provided by the CSP.

8. If the verification is negative, the uCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

9. If the verification is positive, the uCDN computes a Signed URI that is based on unique parameters of that request and provides it to the end user as the URI to use to further request the content from the dCDN.

10. On receipt of the corresponding content request, the dCDN verifies the signed JWT in the Signed URI using the information provided by the uCDN in the CDNI Metadata.

11. If the verification is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

12. If the verification is positive, the dCDN serves the request and delivers the content.

13. At a later time, the dCDN reports logging events that include URI signing information.

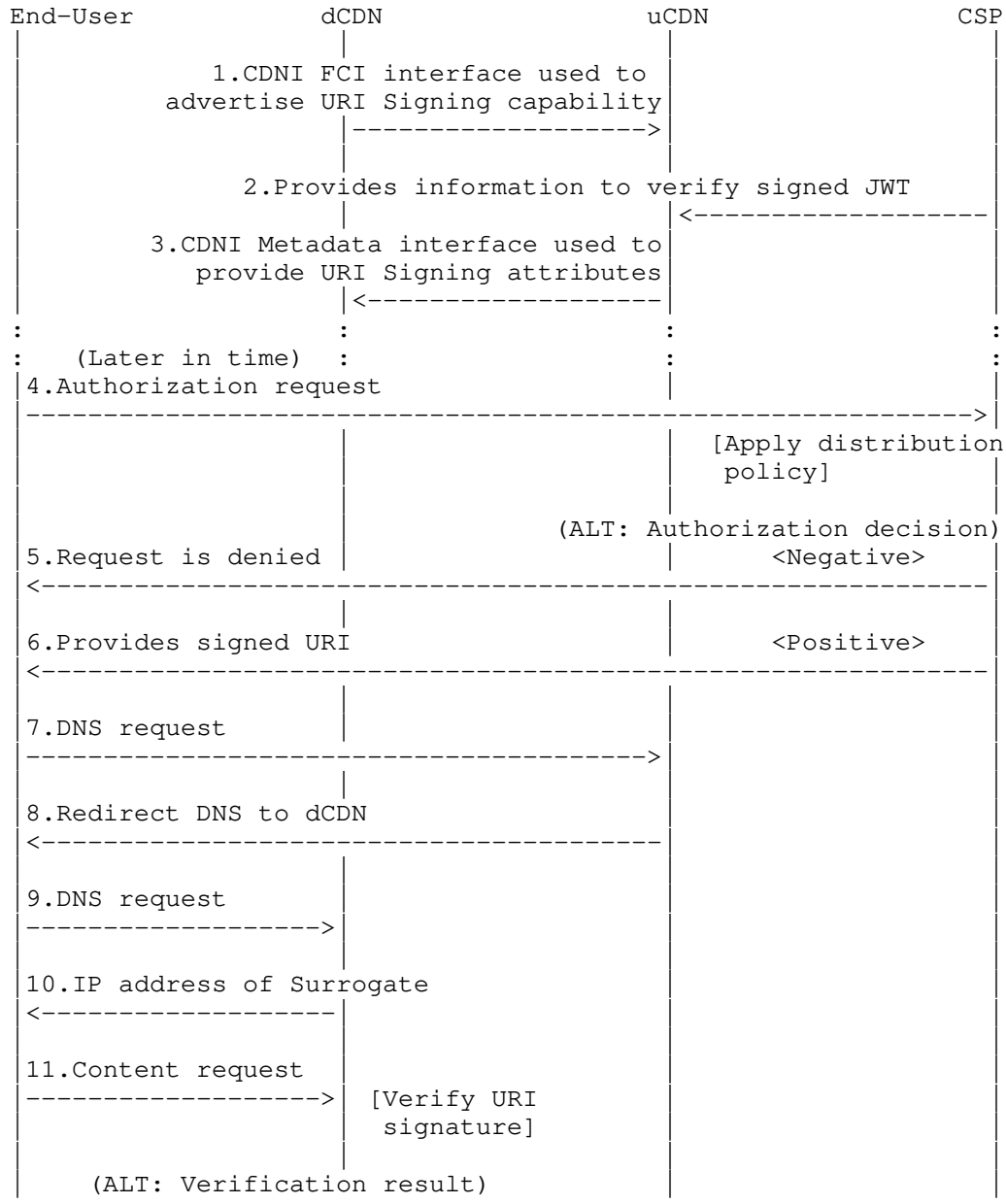
With HTTP-based request routing, URI Signing matches well the general chain of trust model of CDNI both with symmetric and asymmetric keys because the key information only needs to be specific to a pair of adjacent CDNI hops.

5.2. DNS Redirection

For DNS-based request routing, the CSP and uCDN must agree on a trust model appropriate to the security requirements of the CSP's particular content. Use of asymmetric public/private keys allows for unlimited distribution of the public key to dCDNs. However, if a shared secret key is preferred, then the CSP may want to restrict the distribution of the key to a (possibly empty) subset of trusted

dCDNs. Authorized Delivery CDNs need to obtain the key information to verify the Signed URI.

The URI signing method (assuming iterative DNS request routing and a CDN path with two CDNs) includes the following steps.



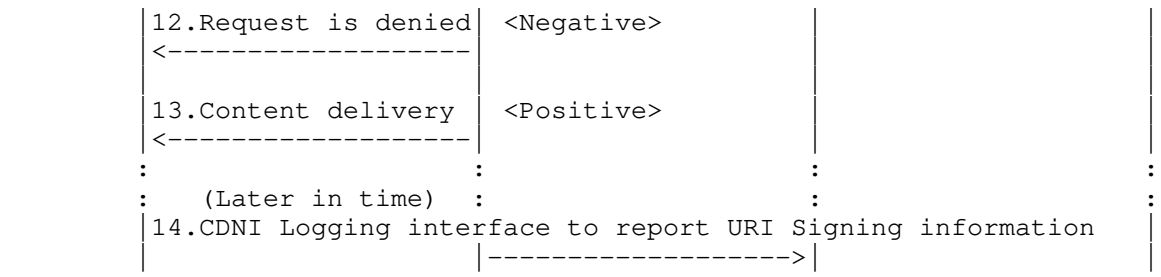


Figure 5: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the dCDN advertises its capabilities including URI Signing support to the uCDN.

2. CSP provides to the uCDN the information needed to verify cryptographic signatures from that CSP. For example, this information may include a key.

3. Using the CDNI Metadata interface, the uCDN communicates to a dCDN the information needed to verify cryptographic signatures from the CSP (e.g., the URI query string parameter name for the URI Signing Package Attribute). In the case of symmetric key, the uCDN checks if the dCDN is allowed by CSP to obtain the shared secret key.

4. When a UA requests a piece of protected content from the CSP, the CSP makes a specific authorization decision for this unique request based on its arbitrary distribution policy.

5. If the authorization decision is negative, the CSP rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.

6. If the authorization decision is positive, the CSP computes a cryptographic signature that is based on unique parameters of that request and includes it in the URI provided to the end user to request the content.

7. End user sends DNS request to the uCDN.

8. On receipt of the DNS request, the uCDN redirects the request to the dCDN.

9. End user sends DNS request to the dCDN.

10. On receipt of the DNS request, the dCDN responds with IP address of one of its Surrogates.

11. On receipt of the corresponding content request, the dCDN verifies the cryptographic signature in the URI using the information provided by the uCDN in the CDNI Metadata.

12. If the verification is negative, the dCDN rejects the request and sends an error code 403 Forbidden in the HTTP response.

13. If the verification is positive, the dCDN serves the request and delivers the content.

14. At a later time, dCDN reports logging events that includes URI signing information.

With DNS-based request routing, URI Signing matches well the general chain of trust model of CDNI when used with asymmetric keys because the only key information that needs to be distributed across multiple, possibly untrusted, CDNI hops is the public key, which is generally not confidential.

With DNS-based request routing, URI Signing does not match well the general chain of trust model of CDNI when used with symmetric keys because the symmetric key information needs to be distributed across multiple CDNI hops, to CDNs with which the CSP may not have a trust relationship. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.

6. IANA Considerations

6.1. CDNI Payload Type

This document requests the registration of the following CDNI Payload Type under the IANA "CDNI Payload Types" registry:

Payload Type	Specification
MI.UriSigning	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.1.1. CDNI UriSigning Payload Type

Purpose: The purpose of this payload type is to distinguish UriSigning MI objects (and any associated capability advertisement).

Interface: MI/FCI

Encoding: see Section 4.4

6.2. CDNI Logging Record Type

This document requests the registration of the following CDNI Logging record-type under the IANA "CDNI Logging record-types" registry:

record-types	Reference	Description
cdni_http_request_v2	RFCthis	Extension to CDNI Logging Record version 1 for content delivery using HTTP, to include URI Signing logging fields

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.2.1. CDNI Logging Record Version 2 for HTTP

The "cdni_http_request_v2" record-type supports all of the fields supported by the "cdni_http_request_v1" record-type [RFC7937] plus the two additional fields "s-uri-signing" and "s-uri-signing-deny-reason", registered by this document in Section 6.3. The name, format, field value, and occurence information for the two new fields can be found in Section 4.5 of this document.

6.3. CDNI Logging Field Names

This document requests the registration of the following CDNI Logging fields under the IANA "CDNI Logging Field Names" registry:

Field Name	Reference
s-uri-signing	RFCthis
s-uri-signing-deny-reason	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.4. CDNI URI Signing Verification Code

The IANA is requested to create a new "CDNI URI Signing Verification Code" subregistry, in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Verification Code" namespace defines the valid values associated with the s-uri-signing CDNI Logging Field. The CDNI URI Signing Verification Code is a 3DIGIT value as defined in Section 4.5. Additions to the CDNI URI Signing Verification Code namespace will conform to the "Specification Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be sparse.

Value	Reference	Description
000	RFCthis	No signed JWT verification performed
200	RFCthis	Signed JWT verification performed and verified
400	RFCthis	Signed JWT verification performed and rejected because of incorrect signature
401	RFCthis	Signed JWT verification performed and rejected because of Issuer enforcement
402	RFCthis	Signed JWT verification performed and rejected because of Subject enforcement
403	RFCthis	Signed JWT verification performed and rejected because of Audience enforcement
404	RFCthis	Signed JWT verification performed and rejected because of Expiration Time enforcement
405	RFCthis	Signed JWT verification performed and rejected because of Not Before enforcement
406	RFCthis	Signed JWT verification performed and rejected because of Issued At enforcement
407	RFCthis	Signed JWT verification performed and rejected because of Nonce enforcement
408	RFCthis	Signed JWT verification performed and rejected because of Version enforcement
409	RFCthis	Signed JWT verification performed and rejected because of Critical Extention enforcement
410	RFCthis	Signed JWT verification performed and rejected because of Client IP enforcement
411	RFCthis	Signed JWT verification performed and rejected because of URI Container enforcement
500	RFCthis	Unable to perform signed JWT verification because of malformed URI

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.5. CDNI URI Signing Signed Token Transport

The IANA is requested to create a new "CDNI URI Signing Signed Token Transport" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Signed Token Transport" namespace defines the valid values that may be in the Signed Token Transport (cdnistt) JWT claim. Additions to the Signed Token Transport namespace conform to the "Specification

Required" policy as defined in [RFC8126]. Updates to this subregistry are expected to be sparse.

The following table defines the initial Enforcement Information Elements:

Value	Description	RFC
0	Designates token transport is not enabled	RFCthis
1	Designates token transport via cookie	RFCthis

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

6.6. JSON Web Token Claims Registration

This specification registers the following Claims in the IANA "JSON Web Token Claims" registry [IANA.JWT.Claims] established by [RFC7519].

6.6.1. Registry Contents

- Claim Name: "cdniv"
- Claim Description: CDNI Claim Set Version
- Change Controller: IESG
- Specification Document(s): Section 2.1.8 of [[this specification]]

- Claim Name: "cdnicrit"
- Claim Description: CDNI Critical Claims Set
- Change Controller: IESG
- Specification Document(s): Section 2.1.9 of [[this specification]]

- Claim Name: "cdniip"
- Claim Description: CDNI IP Address
- Change Controller: IESG
- Specification Document(s): Section 2.1.10 of [[this specification]]

- Claim Name: "cdniuc"
- Claim Description: CDNI URI Container
- Change Controller: IESG
- Specification Document(s): Section 2.1.11 of [[this specification]]

- Claim Name: "cdniets"
- Claim Description: CDNI Expiration Time Setting for Signed Token Renewal
- Change Controller: IESG
- Specification Document(s): Section 2.1.12 of [[this specification]]

- Claim Name: "cdnistt"
- Claim Description: CDNI Signed Token Transport Method for Signed Token Renewal
- Change Controller: IESG
- Specification Document(s): Section 2.1.13 of [[this specification]]

- Claim Name: "cdnistd"
- Claim Description: CDNI Signed Token Depth
- Change Controller: IESG
- Specification Document(s): Section 2.1.14 of [[this specification]]

6.7. Expert Review Guidance

Generally speaking, we should determine the registration has a rational justification and does not duplicate a previous registration. Early assignment should be permissible as long as there is a reasonable expectation that the specification will become formalized. Expert Reviewers should be empowered to pass judgements as they see fit, but should follow the guidelines of [RFC8126] Section 5 when reasonable.

7. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of CDNI. The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a CSP being able to authorize every individual request. It should be noted that URI Signing is not a content protection scheme; if a CSP wants to protect the content itself, other mechanisms, such as DRM, are more appropriate.

In general, it holds that the level of protection against illegitimate access can be increased by including more claims in the signed JWT. The current version of this document includes claims for enforcing Issuer, Client IP Address, Not Before time, and Expiration Time, however this list can be extended with other, more complex, attributes that are able to provide some form of protection against some of the vulnerabilities highlighted below.

That said, there are a number of aspects that limit the level of security offered by URI Signing and that anybody implementing URI Signing should be aware of.

- Replay attacks: A (valid) Signed URI may be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window between the Not Before time and Expiration Time attributes, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent a sudden network issues from denying legitimate UAs access to the content. One may also reduce exposure to replay attacks by including a unique one-time access ID via the Nonce attribute (jti claim). Whenever the dCDN receives a request with a given unique ID, it adds that ID to the list of 'used' IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the dCDN can deny the request based on the already-used access ID.

- Illegitimate clients behind a NAT: In cases where there are multiple users behind the same NAT, all users will have the same IP address from the point of view of the dCDN. This results in the dCDN not being able to distinguish between different users based on Client IP Address which can lead to illegitimate users being able to access the content. One way to reduce exposure to this kind of attack is to not only check for Client IP but also for other attributes, e.g., attributes that can be found in HTTP headers.

The shared key between CSP and uCDN may be distributed to dCDNs - including cascaded CDNs. Since this key can be used to legitimately sign a URL for content access authorization, it is important to know the implications of a compromised shared key.

If a shared key usable for signing is compromised, an attacker can use it to perform a denial-of-service attack by forcing the CDN to evaluate prohibitively expensive regular expressions embedded in a cdniuc claim. As a result, compromised keys should be timely revoked in order to prevent exploitation.

8. Privacy

The privacy protection concerns described in CDNI Logging Interface [RFC7937] apply when the client's IP address (cdniip) is embedded in the Signed URI. For this reason, the mechanism described in Section 2 encrypts the Client IP before including it in the URI Signing Package (and thus the URL itself).

9. Acknowledgements

The authors would like to thank the following people for their contributions in reviewing this document and providing feedback: Scott Leibrand, Kevin Ma, Ben Niven-Jenkins, Thierry Magnien, Dan York, Bhaskar Bhupalam, Matt Caulfield, Samuel Rajakumar, Iuniana Oprescu, Leif Hedstrom, Gancho Tenev, Brian Campbell, and Chris Lemmons.

10. Contributors

In addition, the authors would also like to make special mentions for certain people who contributed significant sections to this document.

- Matt Caulfield provided content for the CDNI Metadata Interface section.
- Emmanuel Thomas provided content for HTTP Adaptive Streaming.
- Matt Miller provided consultation on JWT usage as well as code to generate working JWT examples.

11. References

11.1. Normative References

[POSIX.1] "The Open Group Base Specifications Issue 7", IEEE Std 1003.1 2018 Edition, Jan 2018, <http://pubs.opengroup.org/onlinepubs/9699919799/>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7937] Le Faucheur, F., Ed., Bertrand, G., Ed., Oprescu, I., Ed., and R. Peterkofsky, "Content Distribution Network Interconnection (CDNI) Logging Interface", RFC 7937, DOI 10.17487/RFC7937, August 2016, <https://www.rfc-editor.org/info/rfc7937>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

11.2. Informative References

[IANA.JWT.Claims]
IANA, "JSON Web Token Claims",
<http://www.iana.org/assignments/jwt>.

[MPEG-DASH]
ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment format", ISO/IEC 23009-1:2014, Edition 2, 05 2014, <http://www.iso.org/standard/65274.html>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, DOI 10.17487/RFC6707, September 2012, <https://www.rfc-editor.org/info/rfc6707>.

[RFC6983] van Brandenburg, R., van Deventer, O., Le Faucheur, F., and K. Leung, "Models for HTTP-Adaptive-Streaming-Aware Content Distribution Network Interconnection (CDNI)", RFC 6983, DOI 10.17487/RFC6983, July 2013, <https://www.rfc-editor.org/info/rfc6983>.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, Ed., "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, DOI 10.17487/RFC7336, August 2014, <https://www.rfc-editor.org/info/rfc7336>.

[RFC7337] Leung, K., Ed. and Y. Lee, Ed., "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, DOI 10.17487/RFC7337, August 2014, <https://www.rfc-editor.org/info/rfc7337>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7975] Niven-Jenkins, B., Ed. and R. van Brandenburg, Ed., "Request Routing Redirection Interface for Content Delivery Network (CDN) Interconnection", RFC 7975, DOI 10.17487/RFC7975, October 2016, <https://www.rfc-editor.org/info/rfc7975>.

[RFC8008] Seedorf, J., Peterson, J., Previdi, S., van Brandenburg, R., and K. Ma, "Content Delivery Network Interconnection (CDNI) Request Routing: Footprint and Capabilities Semantics", RFC 8008, DOI 10.17487/RFC8008, December 2016, <https://www.rfc-editor.org/info/rfc8008>.

[RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>.

Appendix A. Signed URI Package Example

This section contains three examples of token usage: a simple example with only the required claim present, a complex example which demonstrates the full JWT claims set, including an encrypted Client IP (cdniip), and one that uses a Signed Token Renewal.

Note: All of the examples have whitespace added to improve formatting and readability, but are not present in the generated content.

All examples use the following JWK Set [RFC7517]:

```
{ "keys": [
  {
    "kty": "EC",
    "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWkldueaImf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S4O7dzB6I4hTiCUvmxCI6FuxWba1xYBlLSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA"
  },
  {
    "kty": "EC",
    "kid": "P5UpOv0eMq1wcxLf7WxIg09JdSYGYFDOWkldueaImf0",
    "use": "sig",
    "alg": "ES256",
    "crv": "P-256",
    "x": "be807S4O7dzB6I4hTiCUvmxCI6FuxWba1xYBlLSSsZ8",
    "y": "rOGC4vI69g-WF9AGEVI37sNNwbjIzBxSjLvIL7f3RBA",
    "d": "yaowezrCLTU6yIwUL5RQw67cHgvZeMTLVZXjUGb1A1M"
  },
  {
    "kty": "oct",
    "kid": "f-WbjxBC3dPuI3d24kP2hfvos7Qz688UTi6aB0hN998",
    "use": "enc",
    "alg": "A128GCM",
    "k": "4uFxxV7fhNmrtiah2d1fFg"
  }
]}
```

Note: They are the public signing key, the private signing key, and the shared secret enctyption key, respectively. The public and private signing keys have the same fingerprint and only vary by the 'd' parameter that is missing from the public signing key.

A.1. Simple Example

This example is a simple common usage example containing a minimal subset of claims that the authors find most useful.

The JWT Claim Set before signing:

Note: "sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY" is the URL Segment form ([RFC6920] Section 5) of "http://cdni.example/foo/bar".

```
{
  "exp": 1474243500,
  "iss": "uCDN Inc",
  "cdniuc": "hash:sha-256;2tderfWPa86Ku7YnzW51YUp7dGUjBS_3SW3ELx4hmWY"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJleHAiOjE0NzQyNDM1MDAsImlzcyI6InVDRE4gS
W5jIiwiY2RuaXVjIjoiaGFzaDpzaGEtMjU2OzJ0ZGVyZldQYTg2S3U3WW56VzUxWVV
wN2RHVWpCU18zU1czRUx4NGhtV1kifQ.qzzAB9akC-HoEzQrkOoODWjMC0PEZRrmWz
2rSMcpLtvxyxVodlB2xcpl4J4ABhLLOJzgzL9B39TljTqZApSOpQ
```

A.2. Complex Example

This example uses all fields except for those dealing with Signed Token Renewal, including Client IP (cdniip) and Subject (sub) which are encrpyted. This significantly increases the size of the signed JWT token.

JWE for Client IP (cdniip) of [2001:db8::1/32]:

```
eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4QkMzZFB1ST
NkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..SuzoOnfg-GVh-BOc.wQ9iS
R1sTj-A04CiDmvcgg.9Ts_cIEUw6Yc6U5HaH1UPQ
```

JWE for Subject (sub) of "UserToken":

```
eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4QkMzZFB1ST
NkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..XsJ7ySeChORSIojp.R1U8E
SGU2NnW.DWR8pTbeCwQZca6SitfX_g
```

The JWT Claim Set before signing:

```
{
  "aud": "dCDN LLC",
  "sub": "eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XYmp4
QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..XsJ7ySeChORS
Iojp.R1U8ESGU2NnW.DWR8pTbeCwQZca6SitfX_g",
  "cdniip": "eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiZGlyIiwia2lkIjoiZi1XY
mp4QkMzZFB1STNkMjRrUDJoZnZvczdRejY4OFVUaTZhQjBoTjk5OCJ9..SuzoOnfg-
GVh-BOc.wQ9iSR1sTj-A04CiDmvcgg.9Ts_cIEUw6Yc6U5HaH1UPQ",
  "cdniv": 1,
  "exp": 1474243500,
  "iat": 1474243200,
  "iss": "uCDN Inc",
  "jti": "5DAafLhZAfhsbe",
  "nbf": 1474243200,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.png"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJhdWQiOiJkQ0ROIExMQyIsInN1YiI6ImV5Smxib
U1pT2lKQk1USTRSME5OSWl3aVlXeG5Jam9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA
0UWtNelpGQjFTVE5rTWpSclVESm9ablp2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T
0NKOS4uWHNKN3lTZUNoT1JTSW9qcC5SMVU4RVNHVTJOblcuRFdSOHBUYmVDd1FaY2E
2U2l0ZlhfZyIsImNkbmlpcCI6ImV5SmxibU1pT2lKQk1USTRSME5OSWl3aVlXeG5Ja
m9pWkdseUlpd2lhMmxrSWpvaVppMVhZbXA0UWtNelpGQjFTVE5rTWpSclVESm9ablp
2Y3pkUmVqWTRPRlZVYVRaaFFqQm9Uams1T0NKOS4uU3V6b09uZmctR1ZoLUJPYy53U
TlpU1Ixc1RqLUEwNENpRG12Y2dnLjlUc19jSUVVdzZZYzZVNUhhSDFVUFEiLCJjZG5
pdiI6MSwiZXhwIjoxNDc0MjQzNTAwLCJpYXQiOjE0NzQyNDMyMDAsImlzcyI6InVDR
E4gSW5jIiwianRpIjoiNURBYWZMaFpBZmhzYmUiLCJuYmYiOjE0NzQyNDMyMDAsImN
kbml1YyI6InJlZ2V4Omh0dHA6Ly9jZG5pXFwuZXhhbXBsZS9mb28vYmFyL1swLTlde
zN9XFwucG5nIn0.XEi1NeP8Lzh6ECcbp6EoqYlnJGikaGp6F3lIJ7ZJt3bim6tOtuD
pCQxmEQxobzIpWOCNdpB8kvxM_s95brKjNQ
```

A.3. Signed Token Renewal Example

This example uses fields for Signed Token Renewal.

The JWT Claim Set before signing:

```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1474243500,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.ts"
}
```

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiY2Rua
XN0ZCI6MiwiZXhwIjoxNDc0MjQzNTAwLCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2R
uaVxcLmV4YW1wbGUvZm9vL2Jhci9bMC05XXszfVxcLnRzIn0.wsSvwxY8mtRax7HK_
dro_l6m-mM-HYdeaUoTSgVS5XTIhXBsCPsYQncsradmgmOWHDDOxsSMVVTjHe5E5YH
ZlQ

Once the server verifies the signed JWT it will return a new signed JWT with an updated expiry time (exp) as shown below. Note the expiry time is increased by the expiration time setting (cdniets) value.

The JWT Claim Set before signing:

```
{
  "cdniets": 30,
  "cdnistt": 1,
  "cdnistd": 2,
  "exp": 1474243530,
  "cdniuc": "regex:http://cdni\\.example/foo/bar/[0-9]{3}\\.ts"
}
```

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiY2Rua
XN0ZCI6MiwiZXhwIjoxNDc0MjQzNTMwLCJjZG5pdWMiOiJyZWdleDpodHRwOi8vY2R
uaVxcLmV4YW1wbGUvZm9vL2Jhci9bMC05XXszfVxcLnRzIn0.SITeoIVZ8-yeE_GBV
jYEo1P2LN-EId1gEJ6baR3Au7Dzh2o_O7LhH3k6wHY081sYMdXHucB0P5ocp-r7gqe
ruQ

Authors' Addresses

Ray van Brandenburg
Tiledmedia
Anna van Buerenplein 1
Den Haag 2595DA
The Netherlands

Phone: +31 88 866 7000
Email: ray@tiledmedia.com

Kent Leung
Cisco Systems, Inc.
3625 Cisco Way
San Jose, CA 95134
United States

Phone: +1 408 526 5030
Email: kleung@cisco.com

Phil Sorber
Apple, Inc.
1800 Wazee Street
Suite 410
Denver, CO 80202
United States

Email: sorber@apple.com