Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

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

October 30, 2017

CDNI extensions for HTTPS delegation
draft-fieau-cdni-interfaces-https-delegation-02

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 a uCDN to a 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 May 3, 2018.

Table of Contents

1. Introduction

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

Several delegation methods are currently proposed within different IETF working groups (refer to [I-D.fieau-cdni-https-delegation] for an overview of delegation works ongoing at the IETF). They specify different methods for provisioning HTTPS delivery credentials.

This document proposes an extension to the CDNI control / Triggers and Metadata interfaces to setup HTTPS delegation between an uCDN and dCDN. Furthermore, it includes a proposal of registry to enable the adding of new methods in the future.

Section 2 is about terminology used in this document. Section 3 presents delegation methods specified at the IETF. Section 4 introduces delegation metadata in CDNI. Section 5 addresses the delegation methods objects. Section 6 describes simple data types.

Section 7 is about an IANA registry for delegation methods.
Section 8 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], CDNI Control interface / Triggers [RFC8007] and Logging interface [RFC7937].

3. Known delegation methods

A few methods are currently being proposed at the IETF to handle delegation of HTTPS delivery between entities, refer to [I-D.fieau-cdni-https-delegation].

Regading the existing delegation methods, we need a common framework in CDNI that provides new requirements on the CDNI interfaces.

This document considers the following methods supporting HTTPS delegation. It may be used between two or more CDNs with applicable interface support following the CDNI framework, such as the CI/ Triggers and Metadata Interface:

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

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

4. Specifying Delegation metadata

Two metadata models for enforcing delegation in CDNI between two entities are suggested in this document:

- New standalone object: SecureDelegation

- Extension to current CDNI metadata model

4.1. SecureDelegation object definition

This section presents an alternative to the approach presented in section 5.1. The section proposes to specify a new metadata object, SecureDelegation, dedicated to provide delegation information between two entities. It aims at fully describing a secured delegation between an uCDN and dCDN by indicating the delegated domain, the start and end duration of a delegation, and the delegation method used.

property: delegateddomains

type: Array

Description: List of delegated hostname indicated by an HostMatch object as defined in RFC8006 section 4.3.3. This value should match the SAN value in certificates.

property: pathpatterns

type: Array

Description: List that contains PathPattern objects with a path to match against a resource's URI path in order to trigger the delegation. It is described in RFC8006, 4.1.4.

property: timewindow

type: TimeWindow

Description: Describes delegation start and end times. Timewindow is defined in RFC8006 section 4.2.

Property: supporteddelegationmethods

type: Array

Description: List of delegation method(s) types that are enabled between a uCDN and a dCDN (ex. "MI.SubcertsDelegationMethod", "MI.AcmeStarDelegationMethod", etc.), as defined in the next section.

As an example: a SecureDelegation object (which contains a TimeWindow, SupportedDelegationMethods and a HostMatch metadata) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

```
 SecureDelegation object:
 {
         "generic-metadata-type": "MI.SecureDelegation",
         "generic-metadata-value":
         {
           "timewindow": {start: 946717200, end: 946746000},
           "supporteddelegationmethods": ["MI.AcmeStarDelegationMethod",
         "MI.SubcertsDelegationMethod"],
           "pathpatterns": [{
               "pattern": "/movies/*",
               "case-sensitive": true
           }],
           "delegatedDomains": ["www.origin.com"]
         }
 }
```

Such an object shall be conveyed over the CDNI metadata interface.

4.2. Extension to the current CDNI metadata model

This approach consists of reusing the current metadata model by adding delegation information, like the aforementioned "supportedDelegationMethod" property.

Example:

As an example, the PathMatch object can reference a path-metadata that would point 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"
 }
}
```

PathMetaData Object related to /movie/*

```
PathMetadata:
{
 "metadata": [
  {
  "generic-metadata-type": "MI.TimeWindowACL",
  "generic-metadata-value": {
   "times": [
    "windows": [
    {
     "start": "1213948800",
     "end": "1478047392"
    }
   ],
   "action": "allow",
  }},
  {
  "generic-metadata-type": "MI.SecureDelegation"
  "generic-metadata-type": {
   "supporteddelegationmethods ": ["MI.AcmeStarDelegationMethod"],
  }
  }
 ]
}
```

The existence of the "MI.SecureDelegation" object in a PathMetaData Object shall enable the use of one of the supported Methods, chosen by the delegate. See next section for more details about delegation methods metadata specification.

5. Delegation methods

This section defines the delegation methods objects metadata. Each object of the section consists in defining metadata related to the following steps:

- Bootstrapping: bootstrapping a secured delegation consists in providing the dCDN with enough parameters to set it up, e.g. ACME servers, Key Servers, etc..

- Credential renewal: In case of certificates based approaches, [I-D.rescorla-tls-subcerts] and [I-D.ietf-acme-star], there is a need in CDNI to periodically provision and update credentials (certificates or private keys) on the dCDNs for a given delegated domain.

- Expiration/Revocation: expiration of delegation can occur for multiple reasons: changes in delegation rights, delegation validity is over. In [I-D.rescorla-tls-subcerts] or [I-D.ietf-acme-star] approaches, the uCDN may implicitly enforce revocation and will prevent any dCDN to renew certificates, or access credentials, when delegation is expired.

- Logging: Regarding logging aspects, we consider to log usages and errors related to a delegated domain. As an example, CDNI logs include: supported delegation method(s), credentials renewal requests, credential revocation notice, mutual agreement for selected credential method to use, credentials download status for a specific domain, as well as errors, related to credentials transfer, or crypto aspects such as bad cypher suite supports, revoked delegations, etc.

5.1. 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] and [I-D.nir-saag-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-time certificate bound to the master certificate.

Property: starproxy

Type: Endpoint

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

Property: acmeserver

Type: Endpoint

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

Property: credentialslocationuri

Type: Link

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

Property: periodicity

Type: Periodicity

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

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
    }
}
```

5.2. SubcertsDelegationMethod object

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

As expressed in [I-D.rescorla-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 to the end-user client application, instead of its own certificate.

Property: credentialsdelegatingentity

Type: Endpoint

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

Property: credentialrecipiententity

Type: Endpoint

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

Property: credentialslocationuri

Type: Link

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

Property: periodicity

Type: Periodicity

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

As an example, AcmeStarDelegationMethod object could express the Acme-Star-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
    }
}
```

6. Metadata Simple Data Type Descriptions

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

6.1. Periodicity

A time value expressed in seconds to indicate a periodicity.

Type: Integer

7. 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.SecureDelegation	TBD
MI.AcmeStarDelegationMethod	TBD
MI.SubCertDelegationMethod	TBD
...	

7.1. CDNI MI SecureDelegation Payload Type

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

Interface: MI/FCI

Encoding: see Section 5.1

7.2. 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 5.1

7.3. 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 5.2

8. Security considerations

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

9. References

9.1. Normative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[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>.

9.2. Informative References

[I-D.fieau-cdni-https-delegation] Fieau, F., Emile, S., and S. Mishra, "HTTPS delegation in CDNI", draft-fieau-cdni-https-delegation-02 (work in progress), July 2017.

[I-D.ietf-acme-star] Sheffer, Y., Lopez, D., Dios, O., Pastor, A., and T. Fossati, "Use of Short-Term, Automatically-Renewed (STAR) Certificates to Delegate Authority over Web Sites", draft-ietf-acme-star-00 (work in progress), June 2017.

[I-D.mglt-lurk-tls] Migault, D., "LURK Protocol for TLS/DTLS1.2 version 1.0", draft-mglt-lurk-tls-01 (work in progress), March 2017.

[I-D.nir-saag-star] Nir, Y., Fossati, T., and Y. Sheffer, "Considerations For Using Short Term Certificates", draft-nir-saag-star-00 (work in progress), October 2017.

[I-D.reschke-http-oob-encoding] Reschke, J. and S. Loreto, "'Out-Of-Band' Content Coding for HTTP", draft-reschke-http-oob-encoding-12 (work in progress), June 2017.

[I-D.rescorla-tls-subcerts] Barnes, R., Iyengar, S., Sullivan, N., and E. Rescorla, "Delegated Credentials for TLS", draft-rescorla-tls-subcerts-02 (work in progress), October 2017.

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: May 3, 2018

O. Finkelman
Qwilt
S. Mishra
Verizon

October 30, 2017

CDNI SVA Extensions
draft-finkelman-cdni-sva-extensions-00

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).

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 May 3, 2018.

Copyright Notice

Table of Contents

1. Introduction

In this document, we describe the different use cases of Open Caching and the interface and functionality extensions they require, compared to the existing CDNI RFCs. 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 Logging interface is described in [RFC7937].

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:

- SVA - Streaming Video Alliance.
- OC - SVA Open Caching.
- RR - Request Router.
- CP - Content Provider.

2. Request routing

This section lists extensions required by request routing features.

2.1. Request router address

Open Caching uses iterative request redirect as defined in [RFC7336]. In order for the uCDN to redirect to the dCDN it requires a request router address. CDNI RFCs do not specify how the request router address is advertised and suggests it may be passed via a bootstrap protocol / interface, which is currently not defined.

We propose to add the request router address as a capability under the Footprint and Capabilities interface.

Use cases

* Footprint: The dCDN may want to have different RR addresses per footprint. Note that a dCDN may spread across multiple geographies. This makes it easier to route client request to a nearby RR. Though this can be achieved using a single canonical name and geo DNS, that approach has limitations, for example a client may be using third party DNS resolver, making it impossible for the redirector to detect where the client is located.

* Scaling: The dCDN may choose to scale its RR service by deploying more RRs in new locations and advertise them via an updatable interface like the FCI.

Proposal

Advertise request router address in an FCI capability object.

Example FCI.RequestRouterAddress object:

```
{
     "capabilities": [
       {
         "capability-type": "FCI.RequestRouterAddress",
         "capability-value": {
           "address": <endpoint object>
         },
         "footprints": [
           <Footprint objects>
         ]
       }
     ]
}
```

2.2. uCDN fallback address

Open Caching requires that the uCDN should provide a fallback address to the dCDN to be used in cases where the dCDN cannot properly handle the request. To avoid redirect loops, the dCDN would redirect the request back to the uCDN but to a different location than the original uCDN address, the uCDN will not redirect requests coming to that other address.

Use cases

* Failover: A dCDN request router receives a request but has no caches to which it can route the request to. This can happen

in the case of failures, or temporary network overload. In these 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 may resolve to redirect back to uCDN.

Proposal

Add a generic metadata object for fallback address similar to the source metadata.

Example MI.FallbackAddress object:

```
{
     "generic-metadata-type": "MI.FallbackAddress",
     "generic-metadata-value":
       {
         "sources": [
           {
             "endpoints": [
               "fallback-a.service123.ucdn.example",
               "fallback-b.service123.ucdn.example"
               ],
             "protocol": "http/1.1"
           },
           {
             "endpoints": ["origin.service123.example"],
             "protocol": "http/1.1"
           }
         ]
       }
}
```

3. Content management

Open Caching uses the CDNI CI/T [RFC8007] as an interface for content management operations. The basic operations are the ones defined in the RFC (i.e. purge, invalidate, pre-position).

3.1. Content matching rules

RFC8007 provides means to match on full content URL or patterns with wildcards. The Open Caching working group proposes to add two more match rule types.

3.1.1. Regular expresssion

Using regexp one can create more complex rules to match on objects for the cases of invalidation and purge.

Use cases

* Purge: Purging specific content within a specific directory path. In some cases wildcard MAY be used but it can be a constraining or overreaching variable that exposes the assets to purge further than desired.

Proposal

Add content.regexs to trigger specification.

Name: content.regexs

Description: Regexs of content the CI/T Trigger Command applies to.

Value: A JSON array of Regexs represented as JSON strings.

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

3.1.2. Playlist

Using video playlist files, one can trigger an operation that will work on a collection of distinct media files in a representation that is natural for the content provider. A playlist may have several formats, specifically HLS *.m3u8 manifest [RFC8216], MSS *.ismc client manifest, and DASH XML MPD file [ISO/IEC 23009-1:2014].

Use cases

* Pre-position: Pre-position of content requires passing the full list of media files to the dCDN. Passing the manifest instead is a more natural interface for both sides as they are both supposed to be able to properly read and understand the manifest files.

Proposal

Add playlist.urls to trigger specification.

Name: playlist.urls

Description: URLs of video playlist the CI/T Trigger Command applies to.

Value: A JSON array of Regexs represented as JSON strings.

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

3.2. Geo limits

A content operation may apply 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 included or not excluded by the geo limit. Note that the limit here is on the cache location rather than client location.

Use cases

* Pre-position: Certain contracts allow for prepositioning or availability of contract in all regions except for certain excluded regions in the world, including caches. For example, some CPs content cannot ever knowingly touch servers in a specific country, including caches. 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 on. In such cases a purge operation to remove content specifically from that region, is required.

Proposal

Add GEO locations as an option in the trigger specification. We should consider where this locations object is defined. Should this a part of CI/T or there can be a way we can use metadata objects. The generic metadata object MI.LocationAcl has the same syntax, though the meaning is different as the limit here is on caches rather than end user locations.

Example of trigger specification with a geo limit:

```
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"
      ]
  },
  "locations": [
    {
      "action": "allow" / "deny",
      "footprints": [
        {
          "footprint-type": "countrycode",
          "footprint-value": ["us"]
        }
      ]
    }
  ],
  "cdn-path": [ "AS64496:1" ]
}
```

3.3. Scheduled operations

A uCDN may wish to perform content management operation on the dCDN with a defined local time schedule.

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. This requires an interface that directs the dCDN when to pre-position the content; the time frame is local time per area as the off-peak time is also localized.

Proposal

Add an execution time window as an option in the trigger specification.

Example of trigger specification with a schedule limit:

```
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"
      ]
  },
  "time-windows": [
    {
      "time-type": "local" / "UTC",
      "start": "<seconds since UNIX epoch>",
      "end": "<seconds since UNIX epoch>"

    }
  ],
  "cdn-path": [ "AS64496:1" ]
}
```

3.4. Trigger extensibility

There are cases in which some new data has to pass in the trigger which was not thought of in advance. We propose the add a mechanism to the trigger spec which will be similar to the MI generic metadata, allowing parties to easily add more information, that can later be standardized if required.

Use cases

* Purge content by acquisition time: A uCDN finds that due to configuration mistake it has delivered wrong content, in the past two hours. The uCDN would like to instruct the dCDN to invalidate all content that was acquired in the past two hours. However, there is no such primitive in the trigger specification. If this would be a common use case it may require the addition of a new generic trigger spec object that restrict the match to be on content which was acquired in some time spec.

* Pre-position by cache type: The uCDN would like the dCDN to pre-populate some content, but only on a specific layer of the caching network, for example, only on home caches. There is currently no such option in the interface. By using a generic object parties may define such object and implement it between them, and later standardize it, if required.

Proposal

Add trigger extensibility mechanism to the trigger specification.

Example of trigger extension:

```
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": "purge",
    "content.patterns": [
        "https://www.example.com/*"
      ]
  },
  "generic-trigger-spec-type": <type-name>,
  "generic-trigger-spec-value":
    {
      <properties of this object>
    }
}
```

3.5. Capabilties

The capabilities added to the triggers interface are not mandatory to support and are, therefore, best negotiated via the FCI.

Use cases

* Content management operations: Advertise which content operations are supported by the dCDN. CDNI defines three operations (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 mapping types: Advertise which mapping types are supported, for example, if adding content regexp and possibly playlists, not all dCDN would support it. For playlist, advertise which types and versions of protocols are supported, e.g. HLS/DASH/SS, DASH templates.

* Trigger spec objects: Advertise which trigger spec object are supported, for example time-window, geo-limit etc.

Proposal

Define the non-mandatory objects as generic objects, similar to the metadata generic objects, and then the FCI can declare which ones of the trigger spec objects are supported. .

4. Split authentication

Different CDNs and Content Providers apply different access control and authentication of user requests. It is not feasible for a dCDN, or ISP cache layer, to implement every scheme a uCDN may have thought of, and, unfortunately, it is not reasonable to expect that uCDNs and CPs will move from their current implementation to a new standard, any time soon. In some cases, existing implementation also include secrets under NDA; sharing them with a third party dCDN is unlikely to happen. Therefore, we aim to look for a solid, generic solution that keeps the access control, authentication and authorization logic in the origin/uCDN.

Use cases

* URI signing: There are numerous methods in which a CP signs its URIs such that the uCDN can verify the signatures. In most cases, symmetric keys are being used and require some key exchange. Expecting the dCDN caches to implement every method used by commercial CDNs is problematic, and sharing of content provider keys is unlikely.

* Token based authentication: Some CPs and CDNs are using token based client / session authentication. The token is passed either as a URI query parameter or as a cookie. The dCDN / ISP cannot implement the token validation, as it has no knowledge of the identity and validation methods used by the CP / uCDN. Also, if using cookies with HTTP redirect, the cookie will be omitted after the redirect, so a solution for cookie based authentication is necessary.

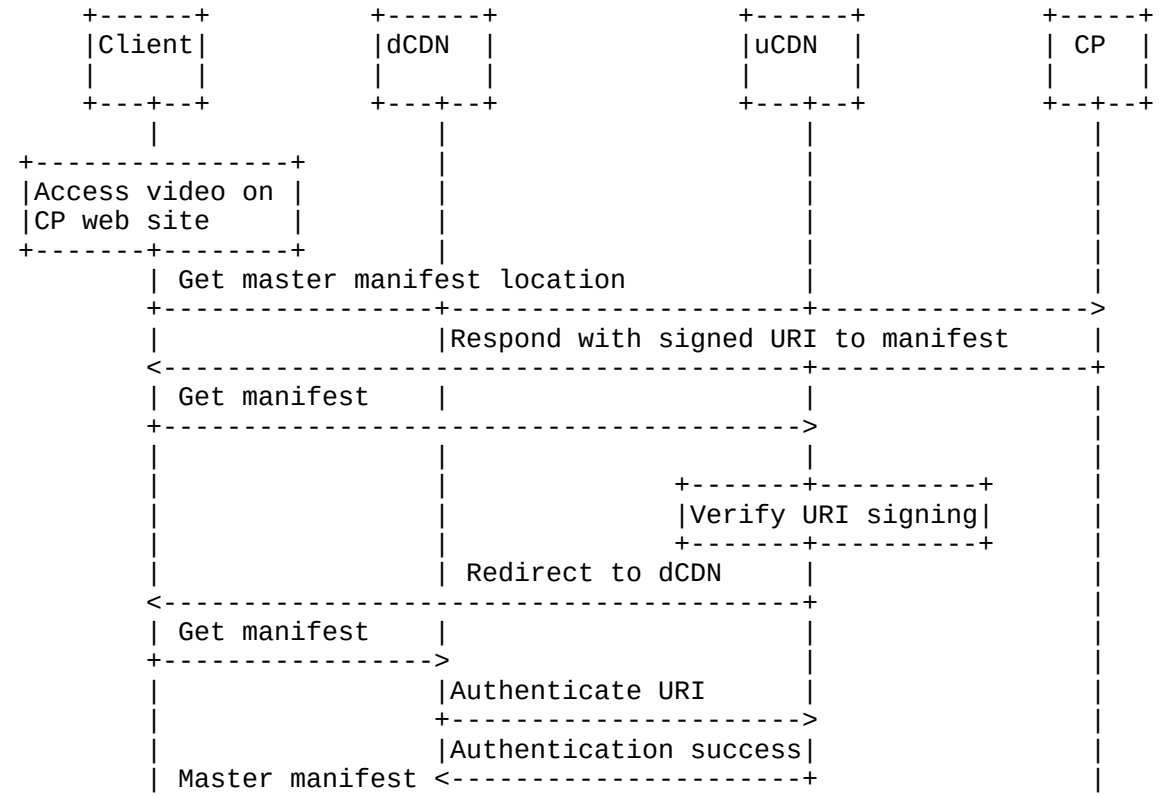
* CORS delegation: CORS may also be a use case of split authentication, see explanation in the CORS delegation section.

Proposal

Split authentication is a mechanism that leverages the fact that video sessions are very long and chunked into very small requests, comparing the overall session time and volume. The dCDN cache relays the authentication verification to the uCDN by sending the uCDN a HEAD request for every new session. The dCDN cache saves the session state for some time and uses it for subsequent requests of the same session.

As this is a general problem when delegating traffic between CDNs, and in-fact, can become a blocker for CDNI deployments. We propose to consider this concept for the general CDNI use case, and draft it for RFC.

The following diagram gives a high level sequence view of the URI signing use case.



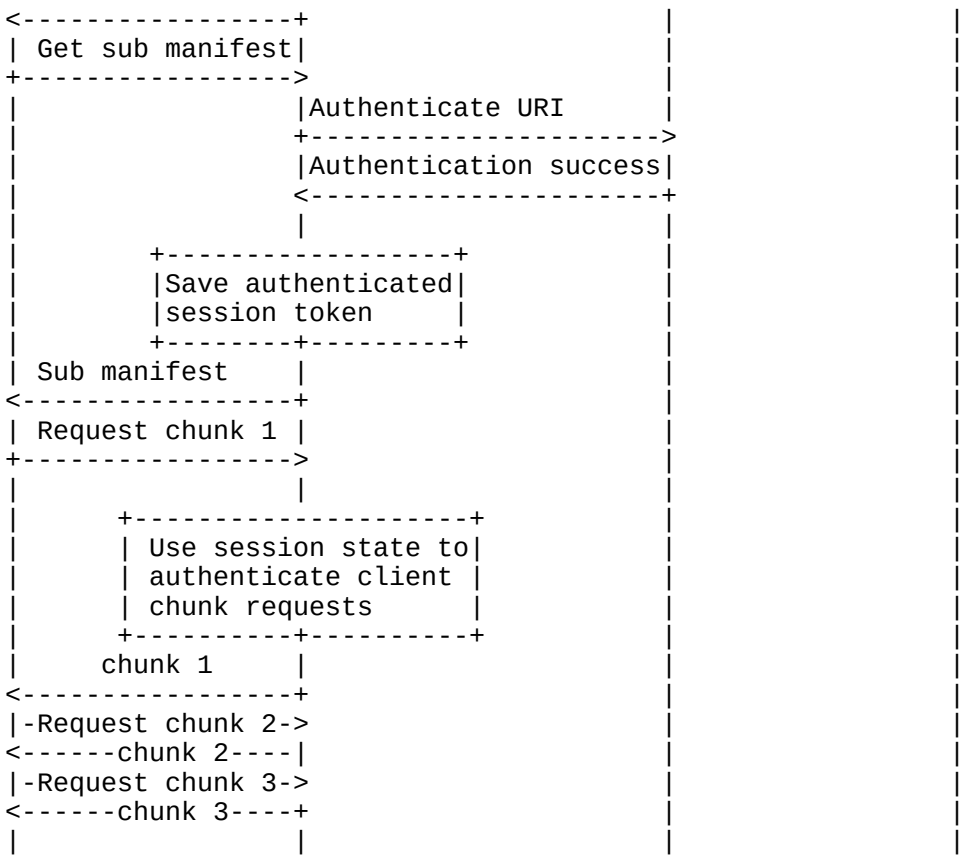


Figure 1

5. CORS delegation

CORS (Cross Origin Resource Sharing) is a mechanism designed to allow a resource from domain A to access other resources in domain B, overriding the same-origin policy. When a uCDN delegate traffic to a dCDN (or ISP) the dCDN is required to comply with the same CORS server behavior the uCDN would have had. For example, if a resource from domain A is accessible for request coming from a resource domain B, but not accessible to requests coming from a resource of domain C, the same logic must be done by the dCDN.

Though CORS can possibly be handled by simply echoing the Origin header value, or *, back to the client, in some cases it is not sufficient, and it also breaks the concept of CORS as an access control mechanism. As proper CORS handling is not possible without a

delegation scheme, the Open Caching working group sees it as an essential part of inter-CDN delegation, and therefore propose to adopt it under CDNI and draft it for CDNI RFC.

Use cases

* A simple use case example is a when resource from Origin: www.video.example.com points to the media file on domain: www.cdn.com. The uCDN is supposed to deliver the content if the Origin is video.example.com otherwise it should be rejected. In this case, for a request header "Origin: www.video.example.com" the CDN should reply with "Access-Control-Allow-Origin: www.video.example.com". OTOH, if the origin is www.video.other.com then the CDN should not allow it by omitting the ACAO header. When delegating the session to a dCDN cache, it should maintain the same behavior.

Proposals

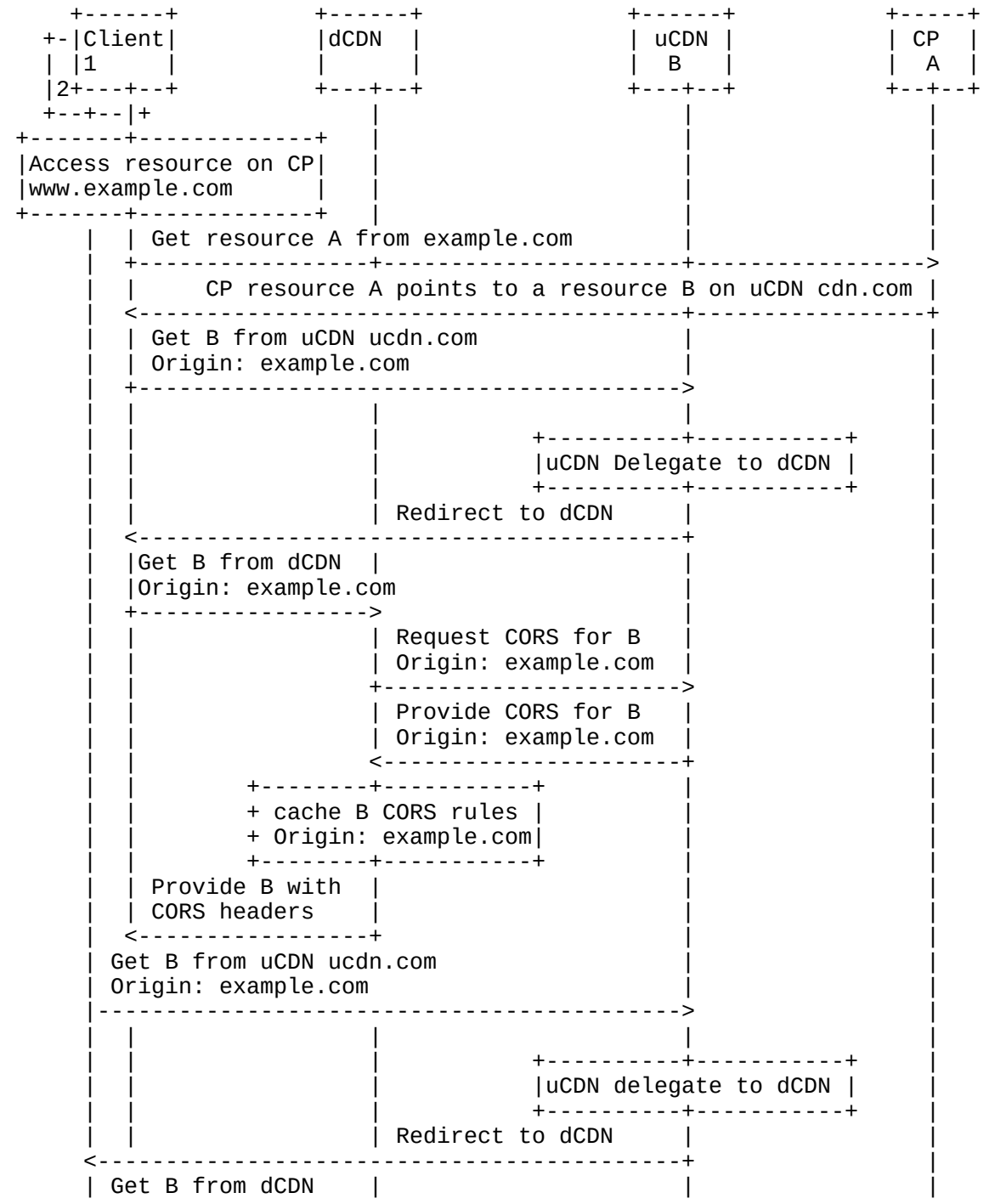
There are several alternatives for the dCDN / ISP cache to learn the allowed origins for a content item.

1. Caching: Caching of CORS headers per content. If the cache receives a request using an origin it does not already approve for that content, the cache sends a HEAD request to the CDN with the client's CORS request headers. The cache saves the response information in a content database and uses it for subsequent requests for the same content. .

2. Metadata: the uCDN can provide the dCDN the metadata referring the content of a specific domain. This metadata holds, for example, all the information required to take CORS decisions at the Open Cache.

3. Split authentication: Using split authentication, the dCDN cache can send the CORS headers to the uCDN in the initial session request, the uCDN responds to the CORS request properly, the dCDN forwards the CORS response to the client and caches it for rest of the client session.

The following diagram gives a high level sequence view of CORS delegation from uCDN to dCDN using the CORS caching alternative.


Client 1
Client 2
dCDN
uCDN B
CP A
Access resource on CP www.example.com
Get resource A from example.com
CP resource A points to a resource B on uCDN cdn.com
Get B from uCDN ucdn.com
Origin: example.com
uCDN Delegate to dCDN
Redirect to dCDN
Get B from dCDN
Origin: example.com
Request CORS for B
Origin: example.com
Provide CORS for B
Origin: example.com
+ cache B CORS rules
+ Origin: example.com
Provide B with
CORS headers
Get B from uCDN ucdn.com
Origin: example.com
uCDN delegate to dCDN
Redirect to dCDN
Get B from dCDN

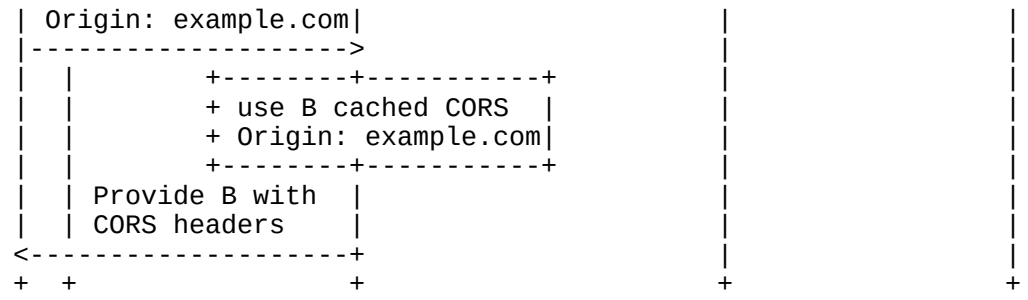


Figure 2

In the above simplified example, we depict the caching alternative for CORS solution.

Client 1 accesses resource A on CP domain example.com. Resource A, refers client 1 to resource B on uCDN ucdn.com. Without delegation, at this points uCDN has to resolve CORS and decide if a resource from example.com is allowed to access a resource at ucdn.com. However, once delegated to dCDN, it becomes the dCDNs duty to resolve it for the client request arrives at the dCDN cache. The dCDN sends a CORS request to the uCDN, for resource B with origin example.com, it then uses the response to respond to client 1, and caches the response. When client 2's request arrives at the dCDN, the required CORS information is already in cache and the dCDN can serve client 2 without reiterating to uCDN.

For simplicity, in this diagram, we have ignored some of the challenges of CORS delegation like preflight requests and "null" origin after HTTP redirect.

6. Logging

This section outlines creation of service delivery logs at the dCDN (ISP) and transmittal of the logs by the dCDN to the uCDN. The key motivation for logging outlined below as compared to CDNI Logging Interface [RFC7937] is the ability for dCDN and uCDN to negotiate and agree on a log transport mechanism.

The logging mechanism provides the flexibility for CDNs to leverage common transport mechanism in-use already. Second, the open caching working group has selected Squid based file format given its wide usage within the CDN environments for access and cache logs, result codes and error messages. As an example, the result codes in squid return both the status code returned by downstream as well as result code indicator such as HIT, MISS, REFRESH_HIT, etc. Between the two statuses, it is easier to discern the delivery status. As an example, if the request was forbidden by the origin, the status field

will likely be MISS/403 or if it is a cache error response, it will be HIT/503. So, leveraging the Squid log already in use within the CDN environment and, equally important, the ability for CDNs to negotiate and agree on a file transport mechanisms, were the key motivations for open caching. These are therefore proposed as complementary extensions to the CDNI Logging Interface [RFC7937].

The sub-sections below explain extensions to the Footprint and Capabilities [RFC8008] and Metadata Interface [RFC8006]. The specific extension includes FCI announcement of supported log file transport types by dCDN and metadata response by uCDN to provision one or more log file types from the list sent by the dCDN.

Use cases

* Transport: Delivery logs are to be supplied by the dCDN to the uCDN via a transport mechanism of choice, supported by both dCDN and uCDN.

* Record format: Log record format is advertised by the dCDN and interpreted correctly by the uCDN. The dCDN in this case shall announce to uCDN one or more transport format that it supports. The uCDN, in turn, will select one format from the potential candidates and set up a provisioning process.

* Log destination: The uCDN configures a log receiving system tied to a specific delivery service it has delegated to a dCDN. The uCDN will provision log destination at its end where it will route the returned logs by delivery service associated with the log file.

The diagram below illustrates the use cases:

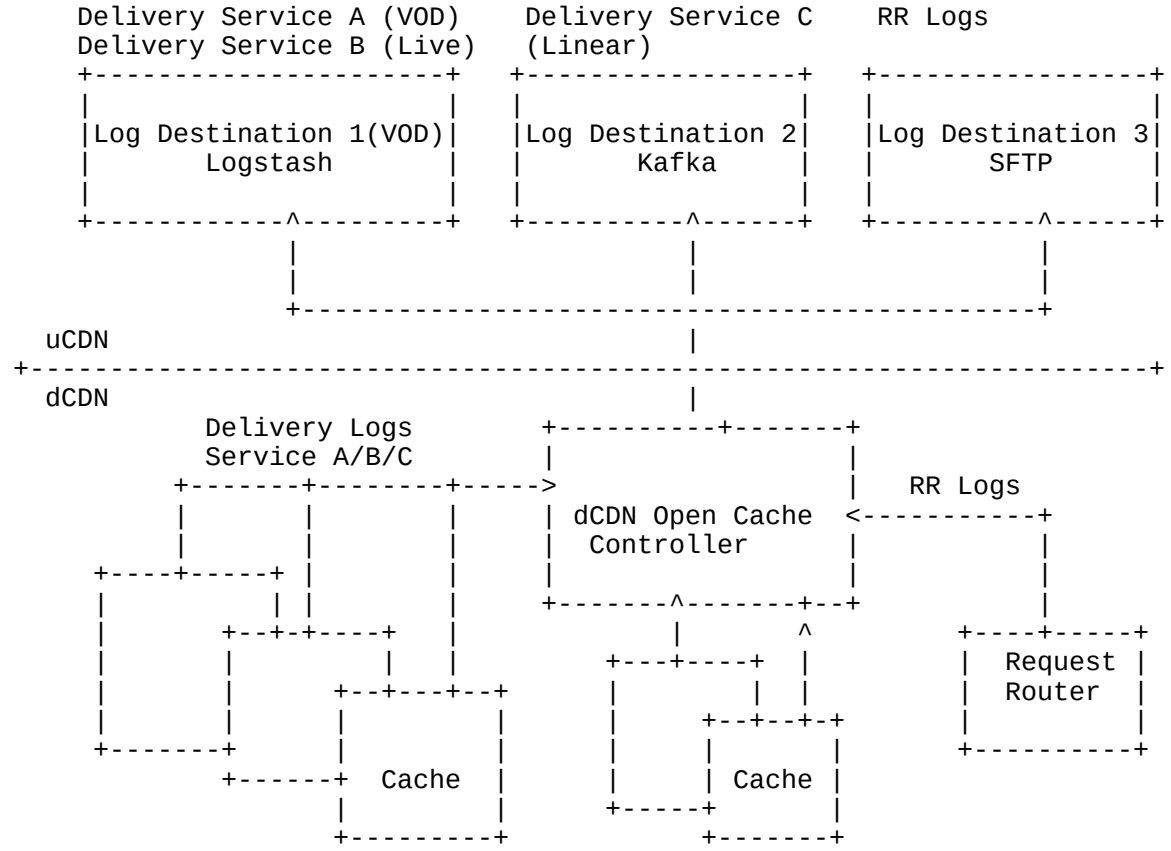


Figure 3

Proposal

Delivery logs are created and then transferred from log producing entities at the dCDN premises (mainly caches and Request Router) to log destinations at the uCDN premises. The dCDN may offload logs from these entities to logging at the dCDN premises to facilitate log transfers, or, logs may be transferred directly from log producing entities to uCDN.

Various transport mechanisms may suit the use case of transferring log data, for example SFTP, HTTP upload, Kafka, Logstash or other methods as per the agreement between a dCDN and a uCDN.

In compliance with the CDNI Footprint and Capabilities Interface, and therefore, as per the above use cases, the dCDN is responsible to advertise supported Logging "record-types", as well as Logging

"fields" which are marked as optional for the s pecified "record-types" as defined by the CDNI "Logging Capability Object".

The CDNI Logging Capability Object is extended to contain additional properties that hold information on record format, such as fields that should be obfuscated by the dCDN. Note that the uCDN can further control field obfuscation when configuring a logging integration.

During provisioning process the dCDN may reject configuration if a selected record format is not available for a selected Log Integration Type.

6.1. FCI extension for Logging

This is a proposal of a Logging Capability object that extends the CDNI "FCI.Logging" object.

The following shows an example of Logging Capability object serialization, for a dCDN that supports the optional fields "hostname" and "cache-key", for the "oc_http_request_v1" record type. The "client-address" field is hashed.

In this example, the logging integration types that are supported are named "kafka" and "logstash"

```
    {
     "capabilities": [
       {
         "capability-type": "FCI.Logging",
         "capability-value": {
           "transport-types": [
             "kafka",
             "logstash"
           ],
           "record-type": "oc_http_request_v1",
           "fields": [
             "hostname",
             "cache-key"
           ],
           "hash-fields": [
             "client-address"
           ]
         },
         "footprints": [
            <footprint-objects>
         ]
       }
     ]
   }
```

6.2. Metadata Interface extension for Logging

This is a proposal of Logging Metadata and Transport Metadata objects that comply with the CDNI "Service Metadata" interface

6.2.1. Logging Configuration object

The following shows an example of Logging Configuration MI.Logging Metadata object serialization, for a logging integration that includes the optional field "hostname" in the log record.

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.Logging",
      "generic-metadata-value": {
        "include-fields": [
          "hostname"
        ]
      },
      "footprints": [
         <footprint-objects>
      ]
    }
  ]
}
```

6.2.2. Transport Configuration object

An initial set of logging transport types and their respective configuration objects should be defined. More types can be added in the future as needed. The following shows an example of Transport Configuration MI.LoggingTransport Metadata object serialization, for a "kafka" logging integration type.

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.LoggingTransport",
      "generic-metadata-value": {
        "type": [
          "kafka",
        ],
        "config":
          <kafka-integration-config-object>
        ]
      },
      "footprints": [
         <footprint-objects>
      ]
    }
  ]
}
```

7. IANA Considerations

7.1. CDNI Payload Types

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

Payload Type	Specification
FCI.RequestRouterAddress	RFCthis
MI.FallbackAddress	RFCthis
MI.Logging	RFCthis
MI.LoggingTransport	RFCthis

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

7.1.1. CDNI FCI RequestRouterAddress Payload Type

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

Interface: FCI

Encoding: see Section 2.1

7.1.2. CDNI MI FallbackAddress Payload Type

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

Interface: MI/FCI

Encoding: see Section 2.2

7.1.3. CDNI MI Logging Payload Type

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

Interface: MI/FCI

Encoding: see Section 6.2.1

7.1.4. CDNI MI LoggingTransport Payload Type

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

Interface: MI/FCI

Encoding: see Section 6.2.2

8. Security Considerations

TBD.

9. Acknowledgements

The authors would like to thank Kevin J. Ma for his guidance and support.

10. Contributors

The authors would like to thank all members of the SVA's Open Caching Working Group for their contribution in support of this document.

11. References

11.1. 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>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[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>.

[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>.

[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>.

[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>.

[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>.

11.2. Informative References

[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>.

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

CDNI
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

R. van Brandenburg
Tiledmedia
K. Leung
Cisco Systems, Inc.
P. Sorber
Comcast Cable Communications

October 30, 2017

URI Signing for CDN Interconnection (CDNI)
draft-ietf-cdni-uri-signing-13

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) [RFC7519] 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 May 3, 2018.

Copyright Notice

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 the CSP's distribution policy.

Specifically, CDNI Framework [RFC7336] states:

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

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

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

This document proposes a method of signing URIs that allows Surrogates in interconnected CDNs to enforce a per-request authorization performed by the CSP. Splitting the role of performing per-request authorization by the CSP and the role of validating 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 actual CSP distribution policy.

The representation of this method is a Signed JSON Web Token (JWT) [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 CDNI Problem Statement [RFC6707].

This document also uses the terminology of 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 signed JWT is REQUIRED.

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 and always contains the signed JWT which is 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 checks to see if the Signed URI is authentic by verifying the signed JWT. If applicable, it checks whether the IP address of the HTTP request matches that in the Signed URI and if the time window is still valid. After these claims are confirmed to be valid, the CDN delivers the content (#4).

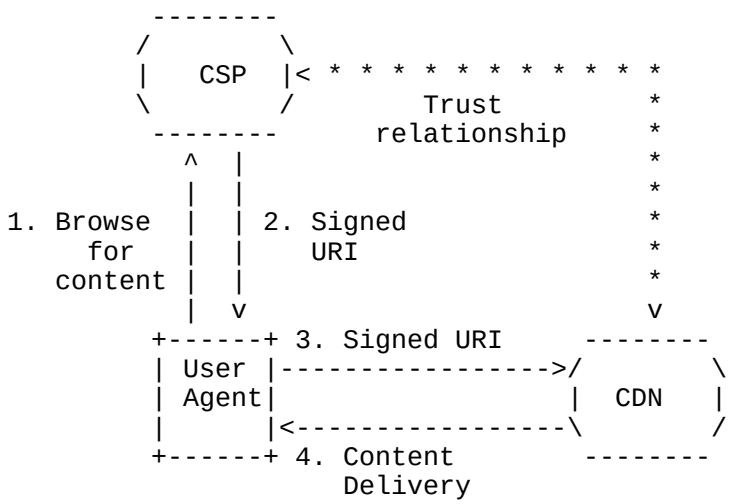


Figure 1: Figure 1: URI Signing in a CDN Environment

1.3. CDNI URI Signing Overview

In a CDNI environment, URI Signing operates the same way in the initial steps #1 and #2 but the later steps involve multiple CDNs in the process of delivering the content. The main difference from the single CDN case is a redirection step between the uCDN and the dCDN. In step #3, 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 (which is 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 validated by the dCDN before delivering the content (#6). This is depicted in the figure below. Note: The CDNI call flows are covered in Detailed URI Signing Operation (Section 5).

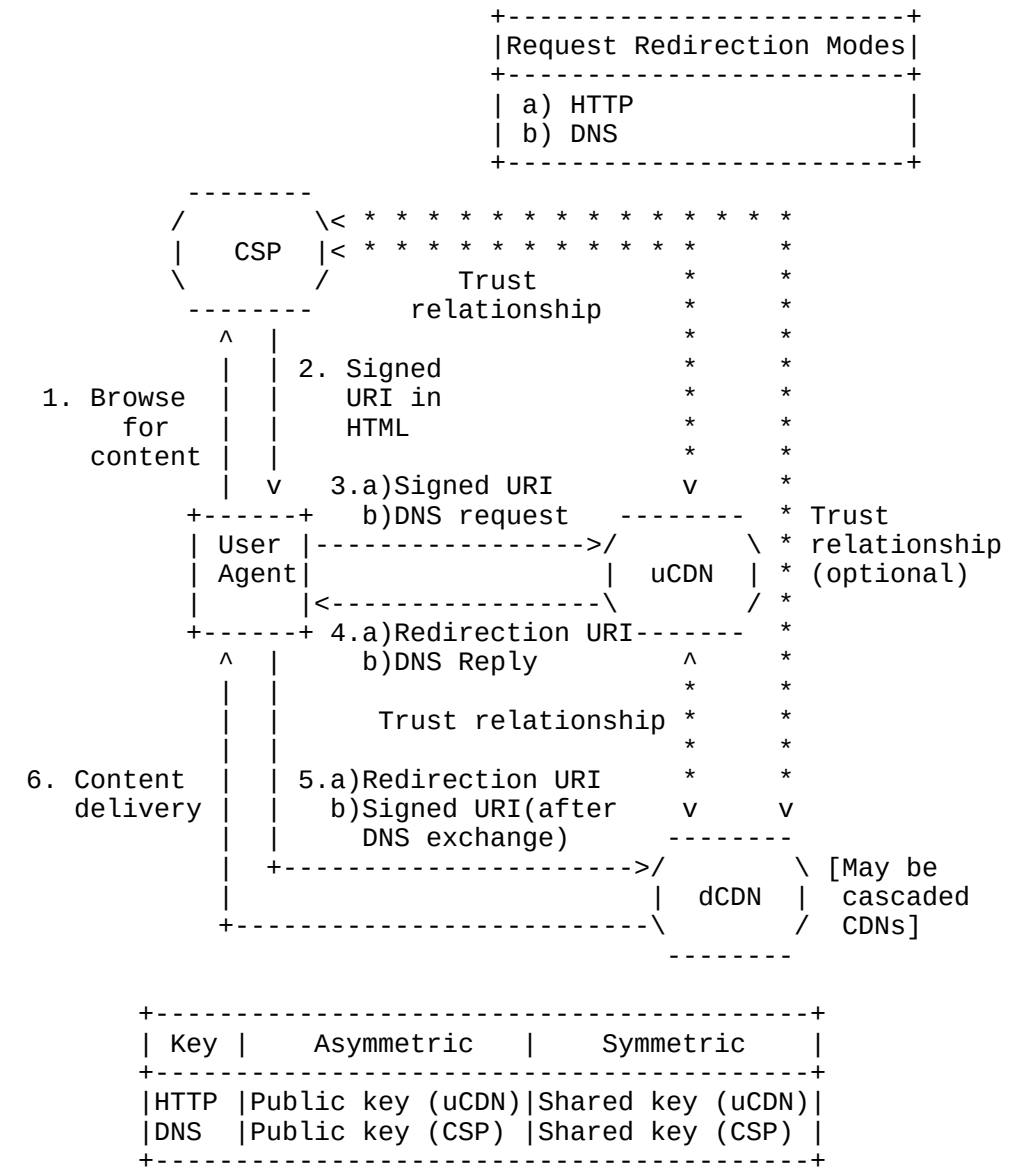


Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, uCDN, and dCDN have direct implications for URI Signing. In the case shown in Figure 2, the CDN that the CSP has a trust relationship with is the uCDN. The delivery of the content may be delegated to the 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., 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., Target CDN URI) provided by the CSP reaches the uCDN. After this URI has been verified to be correct by the uCDN, the uCDN creates and signs a new Redirection URI to redirect the UA to the dCDN. Since this new URI could have a new signed JWT, a new signature can be based around the trust relationship between the uCDN and dCDN, and the relationship between the dCDN and CSP is not relevant. Given the fact that such a relationship between uCDN and dCDN always exists, both asymmetric public/private keys and 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.

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 only known 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 validating entity for validating the Signed URI. Regardless of the type of keys used, the validating entity has to obtain the key (either the public or the symmetric key). There are very different requirements for key distribution (out of scope of this document) with asymmetric keys and with symmetric keys. Key distribution for symmetric keys requires confidentiality to prevent another party from getting access to the key, since it 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 for URI signing) and private keys are kept by the URI signing function.

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, e.g., between a uCDN and a dCDN or between a CSP 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 the UA request: The signed JWT contains a number of claims that can be validated 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).

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. If the signed JWT contains any other claims which the CDN does not understand (i.e., is unable to parse and process), the CDN MUST reject the request.

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 validate 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 validating the signed JWT does not support Issuer validation, 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. URI Container (sub) claim

URI Container (sub) [mandatory] - The semantics in [RFC7519] Section 4.1.2 MUST be followed. Container for holding the URI representation before a URI Signing Package is added. This representation can take one of several forms detailed in Section 2.1.11. If the URI pattern/regex in the signed JWT does not match the URI of the content request, the CDN validating 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.

2.1.3. Client IP (aud) claim

Client IP (aud) [optional] - The semantics in [RFC7519] Section 4.1.3 MUST be followed. IP address, or IP prefix, for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 or canonical text representation for IPv6 addresses [RFC5952]. The request is rejected if sourced from a client outside of the specified IP range. 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 validating the signed JWT does not support Client IP validation, 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. 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.

2.1.4. 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 validate 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 validating the signed JWT does not support Expiry Time validation, or if the Expiry Time in the signed JWT corresponds to a time 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 validate 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 validating the signed JWT does not support Not Before time validation, 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 validate 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 validates that the Nonce in the current JWT has never been used before. If the signed JWT contains a Nonce claim and the CDN validating the signed JWT does not support Nonce storage, 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 a specificiation. This 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 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 validated 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.

2.1.10. 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. This document only defines setting its value to '1', which means the signed JWTs are transported via HTTP Cookies in both directions. Additional values for this claim can be defined in Section 6.4.

2.1.11. URI Container Forms

The URI Container (sub) claim takes one of the following forms. More forms may be added in the future to extend the capabilities.

2.1.11.1. URI Simple Container (uri:)

When prefixed with 'uri:', the string following 'uri:' is the URI that MUST be matched with a simple string match to the requested URI.

2.1.11.2. URI Pattern Container (uri-pattern:)

Prefixed with 'uri-pattern:', this string contains one or more URI Patterns that describes for which content the Signed URI is valid. Each URI Pattern contains an expression to match against the requested URI, to check whether the requested content is allowed to be served. Multiple URI Patterns may be concatenated in a single URI Pattern by separating them with a semi-colon (';') character. Each URI Pattern follows the [RFC3986] URI format, including the '://' that delimits the URI scheme from the hierarchy part. The pattern may include the special literals:

- ';' - separates individual patterns when the string contains multiple URI patterns.
- '*' - matches any sequence of characters, including the empty string.
- '?' - matches exactly one character.
- '$' - used to escape the special literals; MUST be followed by exactly one of ';', '*', '?', or '$'.

The following is an example of a valid URI Pattern:

:///folder/content-83112371/quality_*/segment????.mp4

An example of two concatenated URI Patterns is the following (whitespace is inserted after the ';' for readability and should not be present in the actual representation):

http://*/folder/content-83112371/manifest/*.xml;
http://*/folder/content-83112371/quality_*/segment????.mp4

In order to increase the performance of string parsing the URI Pattern, implementations can check often-used URI Pattern prefixes to quickly check whether certain URI components can be ignored. For example, URI Pattern prefixes '*://*/' or '*://*:*' will be used in case the scheme and authority components of the URI are ignored for purposes of pattern enforcement.

2.1.11.3. URI Regular Expression Container (uri-regex:)

Prefixed with 'uri-regex:', this string is any PCRE [PCRE839] compatible regular expression used to match against the requested URI.

Note: Because '\' has special meaning in JSON [RFC7159] as the escape character within JSON strings, the regular expression character '\' MUST be escaped as '\\'.

An example of a 'uri-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 validating the JWT to ensure its authenticity.

2.1.11.4. URI Hash Container (uri-hash:)

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

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 validation. 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 the vulnerability of 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 scheme defined in this section is based on a mechanism through which subsequent segment requests can be linked together. As part of the JWT validation 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 validated 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 Pattern Container and/or the URI Regular Expression Container scoping mechanisms in the URI Container (sub) 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 depends on the exact segmented protocol and since this document is only concerned with the generation and validation of incoming request, this process 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 folowing section defines a mechanism using HTTP Cookies that allows such UAs to support the concept of renewing signed JWTs without requiring any support on the UA side.

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 a 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 below 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. As an example: 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 validate 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 (Editor’s Note: Is this within the scope of 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 validation 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 must be signed and validated before delivering content. Otherwise, the dCDN does not perform validation, 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 validated.

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. Default is "URISigningPackage".

Property: jwt-header

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

Type: String

Mandatory-to-Specify: No. A jwt-header is not essential for all implementations of URI signing.

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"
    }
}
```

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 validation performed by the Surrogate on the request. The allowed values are:
    - "000" : no signed JWT validation performed
    - "200" : signed JWT validation performed and validated
    - "400" : signed JWT validation performed and rejected because of incorrect signature

+ "401" : signed JWT validation performed and rejected because of Expiration Time enforcement

+ "402" : signed JWT validation performed and rejected because of Client IP enforcement

+ "403" : signed JWT validation performed and rejected because of URI Pattern enforcement

+ "404" : signed JWT validation performed and rejected because of Issuer enforcement

+ "405" : signed JWT validation performed and rejected because of Not Before enforcement

+ "500" : unable to perform signed JWT validation because of malformed URI

* occurrence: there MUST be zero or exactly one instance of this field.

o 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

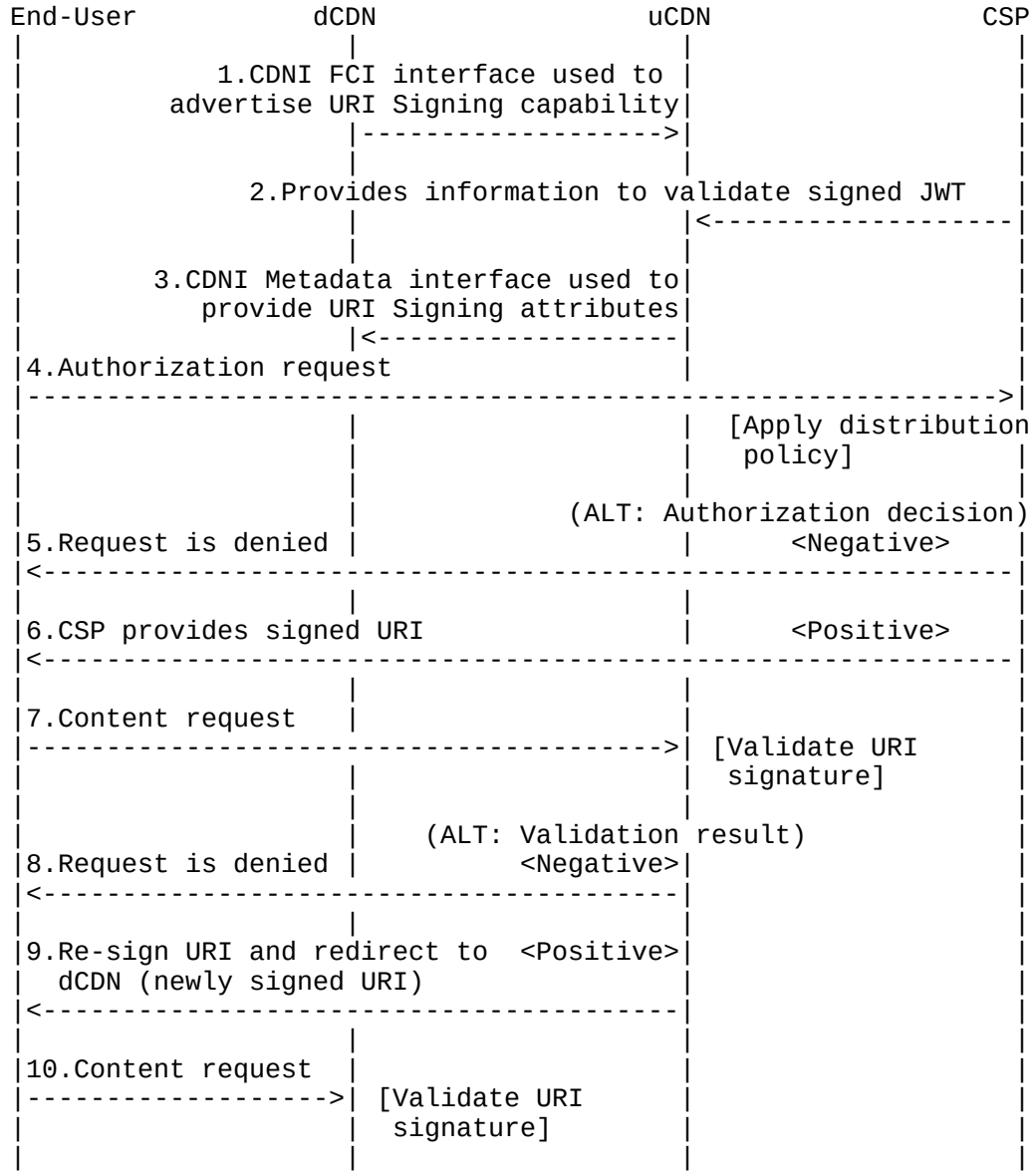
URI Signing supports both HTTP-based and DNS-based request routing. JSON Web Token (JWT) [RFC7519] defines a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a signed JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (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 described below is based on the following steps (assuming HTTP redirection, iterative request routing, and a CDN path with two CDNs). Note that uCDN and uCDN are used exchangeably.



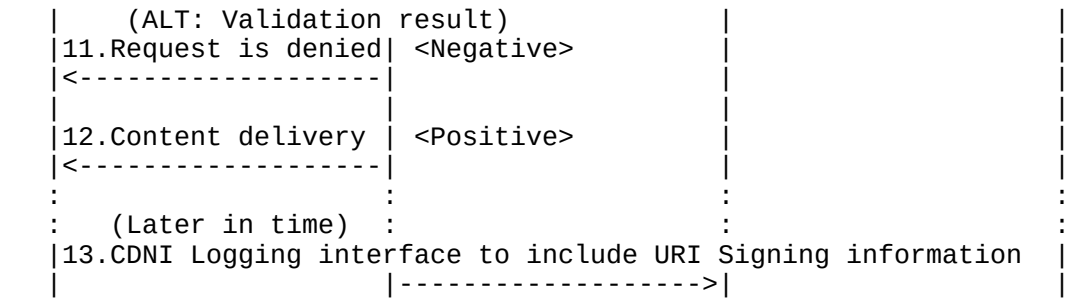


Figure 3: HTTP-based Request Routing with URI Signing

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 validate 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 validate 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 validates the signed JWT in the URI using the information provided by the CSP.

8. If the validation is negative, the uCDN rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.

9. If the validation 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 validates the signed JWT in the Signed URI using the information provided by the uCDN in the CDNI Metadata.

11. If the validation is negative, the dCDN rejects the request and sends an error code (e.g., 403 Forbidden) in the HTTP response.

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

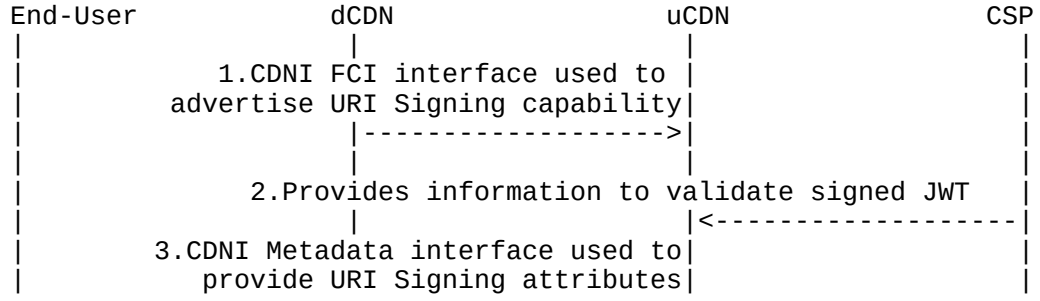
13. At a later time, the dCDN reports logging events that include URI signing information.

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 validate the Signed URI.

The URI signing method described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs).



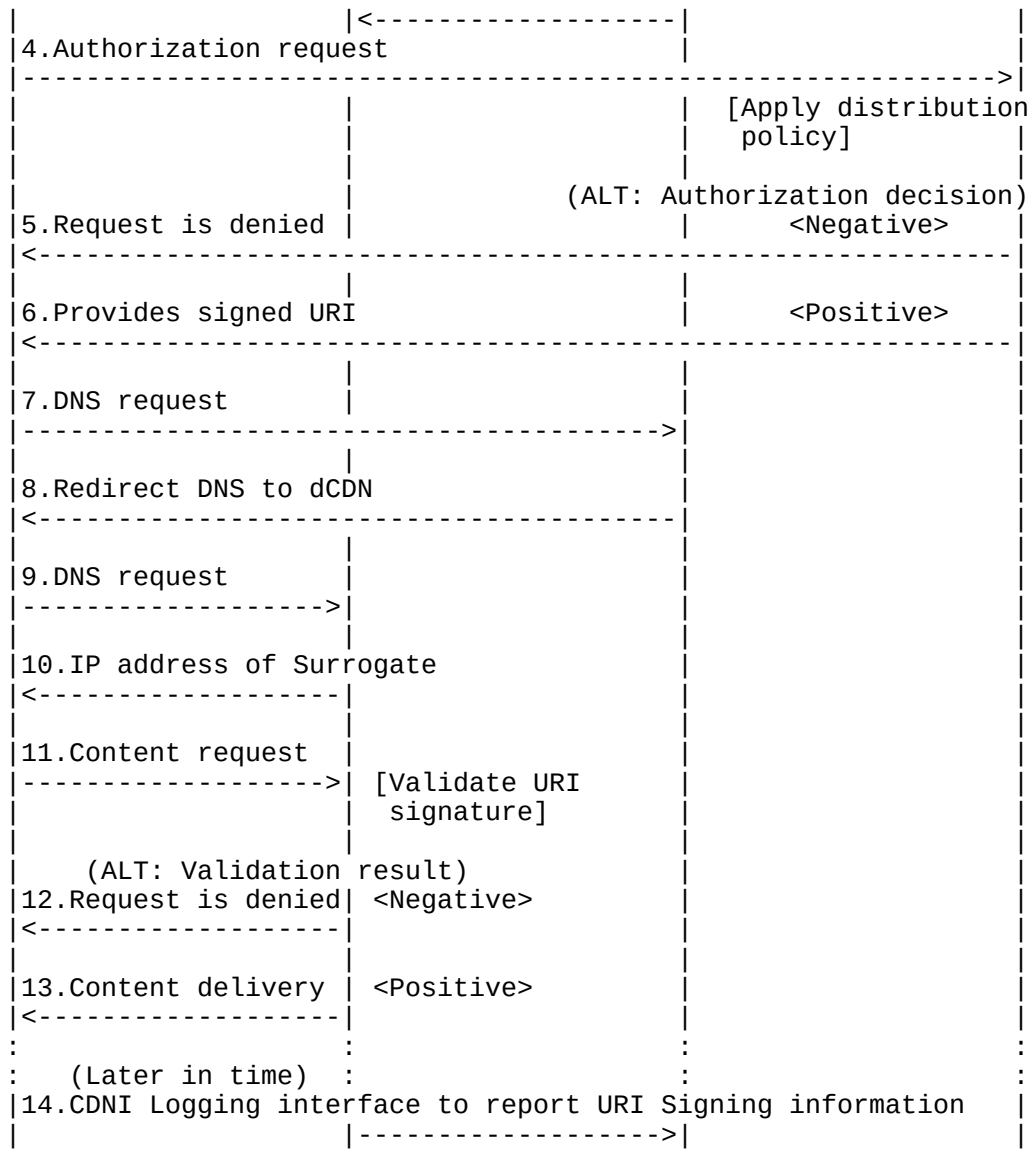


Figure 4: DNS-based Request Routing with URI Signing

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 validate 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 validate 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.

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 validates the cryptographic signature in the URI using the information provided by the uCDN in the CDNI Metadata.

12. If the validation is negative, the dCDN rejects the request and sends an error code (e.g., 403) in the HTTP response.

13. If the validation 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 Type" 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 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 [RFC5226].

The following table defines the initial Enforcement Information Elements:

Value	Description	RFC
1	Designates token transport via cookie	RFCthis

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

[Ed Note: are there any special instructions to the designated expert reviewer?]

6.5. 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].

6.5.1. 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: "cdniets"
- Claim Description: CDNI Expiration Time Setting for Signed Token Renewal
- Change Controller: IESG
- Specification Document(s): Section 2.1.9 of [[this specification]]

- Claim Name: "cdnistt"
- Claim Description: CDNI Signed Token Transport Method for Signed Token Renewal
- Change Controller: IESG
- Specification Document(s): Section 2.1.10 of [[this specification]]

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 preventing 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 the different users based on Client IP Address and 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.

8. Privacy

The privacy protection concerns described in CDNI Logging Interface [RFC7937] apply when the client's IP address (aud) 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

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[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>.

[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>.

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>.

[PCRE839] Hazel, P., "Perl Compatible Regular Expressions", Version 8.39, June 2016, <http://www.pcre.org/>.

[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>.

[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>.

[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>.

[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 (aud), 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 the simplest possible example containing the only required field (sub).

The JWT Claim Set before signing:

```
{
  "sub": "uri:http://cdni.example/foo/bar/baz"
}
```

The signed JWT:

```
eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJzdWIiOiJ1cmk6aHR0cDovL2NkbmkuZXhhbXBsZ
S9mb28vYmFyL2JheiJ9.LTizGd7zCb17Qp_80ClGApLCieRIq3dCjcRckNfLqiJ4BT
GSfVXoDtm0Z6L5Jx4EmwM1w-WkznNajUy11iMAcA
```

A.2. Complex Example

This example uses all optional fields except for those dealing with Signed Token Renewal, including Client IP (aud) which is encrpyted. This significantly increases the size of the signed JWT token.

JWE for client IP (aud) of [2001:db8::1/32]:

```
eyJhbGciOiJkaXIiLCJraWQiOiJmLVdianhCQzNkUHVJM2QyNGtQMmhmdm9zN1F6Nj
g4VVRpNmFCMGhOOTk4IiwiZW5jIjoiQTEyOEdDTSJ9..oGLsnF8fLlFcUXK0.KFfBB
H_FPYFu-RBFBR3rhQ.6_MVaa4t7JiVX2IgUkZ3jA
```

The JWT Claim Set before signing:

```
{
  "aud": "eyJhbGciOiJkaXIiLCJraWQiOiJmLVdianhCQzNkUHVJM2QyNGtQMmhm
dm9zN1F6Njg4VVRpNmFCMGhOOTk4IiwiZW5jIjoiQTEyOEdDTSJ9..oGLsnF8fLlFc
UXK0.KFfBBH_FPYFu-RBFBR3rhQ.6_MVaa4t7JiVX2IgUkZ3jA",
  "cdniv": 1,
  "exp": 1474243500,
  "iat": 1474243200,
  "iss": "uCDN Inc",
  "jti": "5DAafLhZAfhsbe",
  "nbf": 1474243200,
  "sub": "uri-regex:http://cdni\\.example/foo/bar/baz/[0-9]{3}\\.png"
}
```

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJhdWQiOiJleUpoYkdjaU9pSmthWElpTENKcmFXU
WlPaUptTFZkaWFuaENRek5rVUhWSk0yUXlOR3RRTW1obWRtOXpOMUY2TmpnNFZWUnB
ObUZDTUdoT09UazRJaXdpWlc1aklqb2lRVEV5T0VkRFRTSjkuLm9HTHNuRjhmTGxGY
1VYSzAuS0ZmQkJIX0ZQWUZ1LVJCRkJSM3JoUS42X01WYWE0dDdKaVZYMklnVWtaM2p
BIiwiY2RuaXYiOjEsImV4cCI6MTQ3NDI0MzUwMCwiaWF0IjoxNDc0MjQzMjAwLCJpc
3MiOiJ1Q0ROIEluYyIsImp0aSI6IjVEQWFmTGhaQWZoc2JlIiwibmJmIjoxNDc0MjQ
zMjAwLCJzdWIiOiJ1cmktcmVnZXg6aHR0cDovL2NkbmlcXC5leGFtcGxlL2Zvby9iY
XIvYmF6L1swLTldezN9XFwucG5nIn0.r2FiSdfnGRw_RC2roGwh4LEYlfsWGF972-M
f3He_LhGr2_3eRXP0jP_OFPj5NEtTZFY4PX_iD7vPD12_HPDJCQ

A.3. Signed Token Renewal Example

This example uses fields for Signed Token Renewal.

The JWT Claim Set before signing:

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

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiZXhwI
joxNDc0MjQzNTAwLCJzdWIiOiJ1cmktcmVnZXg6aHR0cDovL2NkbmlcXC5leGFtcGx
lL2Zvby9iYXIvYmF6L1swLTldezN9XFwudHMifQ.VYF7Eqk1WhRWdvDVUx5mXDJaS-
r6jbjgiYuwIEAYmbLWsF2dUDohrV70sz7TC09n-oNf_ws4eeH_A6AnvF4FTQ

Once the server validates 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,
  "exp": 1474243530,
  "sub": "uri-regex:http://cdni\\.example/foo/bar/baz/[0-9]{3}\\.ts"
}
```

The signed JWT:

eyJhbGciOiJFUzI1NiIsImtpZCI6IlA1VXBPdjBlTXExd2N4TGY3V3hJZzA5SmRTWU
dZRkRPV2tsZHVlYUltZjAifQ.eyJjZG5pZXRzIjozMCwiY2RuaXN0dCI6MSwiZXhwI
joxNDc0MjQzNTMwLCJzdWIiOiJ1cmktcmVnZXg6aHR0cDovL2NkbmlcXC5leGFtcGx
lL2Zvby9iYXIvYmF6L1swLTldezN9XFwudHMifQ.zzXhYg6b7_8MylW-RS97qZv3hQ
QlYg-TxhC-wigdB4moxlFEjndDvv0EDxl42faQaE39BCHZq7H-DXVpBrhxTw

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
Comcast Cable Communications
1401 Wynkoop Street, Suite 300
Denver, CO 80202
United States

Phone: +1 720 502 3785
Email: phillip_sorber@comcast.com