CDNI
Internet-Draft
Intended status: Standards Track
Expires: December 8, 2016

R. van Brandenburg
TNO

June 6, 2016

URI Signing for HTTP Adaptive Streaming (HAS)
draft-brandenburg-cdni-uri-signing-for-has-03

Abstract

This document defines an extension to the URI Signing mechanism specified in [I-D.ietf-cdni-uri-signing] that allows for URI Signing of content delivered via HTTP Adaptive Streaming protocols such as MPEG DASH or HLS.

The proposed mechanism is applicable to both CDNI as well as single-CDN environments.

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 http://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 December 8, 2016.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-cdni-uri-signing] describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized User Agents (UAs) are able to access specific content, with a Content Service Provider (CSP) being able to authorize every individual request. As noted in [I-D.ietf-cdni-uri-signing], 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.

For content that is delivered via an HTTP Adaptive Streaming (HAS) protocol, such as MPEG DASH or HLS [Editor's Note: Include reference], special provisions need to be made in order to ensure URI Signing can be applied. In general, HAS protocols work by breaking large objects (e.g. videos) into a sequence of small independent chunks. Such chunks are then referenced by a separate manifest file, which either includes a list of URLs to the chunks or specifies an

algorithm through which a User Agent can construct the URLs to the chunks. Requests for chunks 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 chunks.

One method for dealing with this vulnerability would be to include in the manifest itself Signed URIs that point to the individual chunks. 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 HAS protocol and version used. Secondly, it would require the expiration time of the Signed URIs to be valid for at least the full duration of the content described by the manifest. Since it is not uncommon for a manifest file to contain a video item of more than 30 minutes in length, this would require the Signed URIs to be valid for a long time, thereby reducing their effectiveness and that of the URI Signing mechanism in general. For a more detailed analysis of how HAS protocols affect CDNI, see Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983].

The method defined in this document allows CDNs to use URI Signing for HTTP Adaptive Streaming content without having to include the Signed URIs in the manifest files themselves.

1.1. Terminology

This document uses the terminology defined in CDNI Problem Statement [RFC6707] and [I-D.ietf-cdni-uri-signing].

In addition, the following term is used throughout this document:

- Signed Token: A set of URI Signing Information Elements protected by a URI Signature that can be used to retrieve a pre-determined set of resources. It can be communicated through various methods, including the Cookie-based mechanism defined in this document. A Signed Token differs from a Signed URI as defined in [I-D.ietf-cdni-uri-signing] in the sense that it is self-contained and can be communicated outside the context of a particular URL. A sequence of Signed Tokens can be used to form a Signed Token Chain in the case of HTTP Adaptive Streaming content.

- Signed Token Chain: A chain of Signed Tokens that are used for subsequent access to a set of related resources in a CDN. Every time a Signed Token is used to access a particular resource, a new Signed Token is sent along with the resource that can be used to request the next resource in the set. When generating a new

Signed Token in a Signed Token Chain, parameters are carried over from one Signed Token to the next via URI Signing Information Elements.

1.2. URI Signing in a non-CDNI context

While the URI Signing for HTTP Adaptive Streaming scheme 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 scheme 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 section 1.2 of [I-D.ietf-cdni-uri-signing] , for example to allow a CSP that uses different CDNs to only have to implement a single URI Signing mechanism.

2. URI Signing for HAS overview

In order to allow for effective access control of HAS content, the URI signing scheme defined in this document is based on a mechanism through which subsequent chunk requests can be chained together. As part of the URI validation procedure, the CDN can generate a Signed Token that the UA can use to do a subsequent request. More specifically, whenever a UA successfully retrieves a chunk, it receives, in the HTTP 2xx Successful message, a Signed Token that it can use whenever it requests the next chunk. As long as each Signed Token in such a Signed Token Chain is correctly validated before a new one is generated, the chain is not broken and the User Agent can successfully retrieve additional chunks. Given the fact that with HAS protocols, it is usually not possible to determine a priori which chunk will be requested next (i.e. to allow for seeking within the content and for switching to a different quality level), the Signed Token Chain includes a scoping mechanism that allows it to be valid for more than one URL.

In order for this chaining of Signed Tokens to work, it is necessary for a UA to extract the Signed Token from the HTTP 2xx Successful message of an earlier request and use it to retrieve the next chunk. The exact mechanism by which the client does this depends on the exact HAS 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 Tokens, this document does define a mechanism using HTTP Cookies that allows such UAs to support the concept of chained Signed Tokens without requiring any support on the UA side.

3. Signed URI Information Elements

This document defines additional Information Elements beyond those defined in [I-D.ietf-cdni-uri-signing].

3.1. Signature Computation Information Elements

This section specifies additional Information Elements that may be needed to verify and calculate a new Signed Token, in addition to the Signature Computation Information Elements specified in [I-D.ietf-cdni-uri-signing]:

- Expiration Time Setting (ETS) [optional for Signed Token] - An 16-bit unsigned integer (in seconds) used for setting the value of the Expiry Time Information Element in newly generated Signed Tokens. The Expiration Time Setting Information Element MUST NOT be used in a Signed URI as defined in [I-D.ietf-cdni-uri-signing].

- Signed Token Transport (STT) [mandatory for Signed Token] - An 8-bit unsigned integer used for signalling the method through which the Signed Token is transported from the CDN to the UA and vice versa. This document only defines setting the STT Information Element to a value of 1, which means that the Signed Token is transported via a Cookie for both directions.

The Expiration Time Setting Information Element is used to communicate to the CDN to which duration the Expiry Time Information Element should be set whenever a new Signed Token is generated.

The Signed Token Transport Information Element is used to to communicate to the CDN which method to use for transporting the Signed Token to the UA

4. Creating an initial Signed Token

The following procedure defines the algorithm for creating the initial Signed Token of a Signed Token Chain. Note that the process described in this section is only performed for creating the initial Signed Token of a particular Signed Token Chain. Subsequent Signed Tokens forming the same Signed Token Chain are generated as part of the URI Signature Validation process described in Section 7. The creation of the initial Signed Token will typically be done by the CSP the first time a particular UA requests the manifest file. Choosing appropriate values of the Enforcement Information Elements in the initial Signed Token requires some knowledge of the structure of the HTTP Adapative Streaming content that is being requested.

In contrast with the Signed URI defined in [I-D.ietf-cdni-uri-signing] a Signed Token MUST always contain a URI Pattern Container Information Element instead of a Original URI Container Information Element. The URI Pattern Container element is used to convey the set of resources for which the particular Signed Token is valid.

The process of generating a initial Signed Token can be divided into two sets of steps: first, calculating the URI Signature and then, packaging the URI Signature along with the URI Signing Information Elements into a URI Signing Package to construct a Signed Token and appending the Signed Token to the message. Note it is possible to use some other algorithm and implementation as long as the same result is achieved. An example for the Original URI, "http://example.com/folder/content-83112371/manifest.xml", is used to clarify the steps.

Note that although the URI Signing for HAS mechanism defined in this document uses most of the Information Elements defined in [I-D.ietf-cdni-uri-signing] and is fully compatible with it, to make it easier for CDNs to distinguish between Signed Tokens and the Signed URIs specified in [I-D.ietf-cdni-uri-signing], the URI Signing Version field is set to ’2’ when Signed Token are used.

4.1. Calculating the URI Signature (initial Signed Token)

Calculate the URI Signature for use with a Signed Token by following the procedure below.

1. Create an empty buffer for constructing the Signed Token and performing the operations below.

2. Place the string "VER=2" in the buffer.

3. If time window enforcement is needed, perform this step.

    A. Append an "&" character to the buffer. Append the string "ET=".

    B. Get the current time in seconds since epoch (as an integer). Add the validity time (in seconds) of the initial Signed Token as an integer.

    C. Convert this integer to a string and append to the buffer.

    D. Append an "&" character to the buffer. Append the string "ETS=".

E. Append the Expiration Time Setting (in seconds) in the form of a string to the message. Note: the length of the Expiration Time Setting should be appropriate given the segment duration of the HTTP Adaptive Streaming content in question. As an example, if the segment duration is 10 seconds, the Expiration Time Setting should be at minimum 10 seconds, and preferably a bit more.

4. If client IP enforcement is needed, perform this step.

   A. Skip this step if the Client IP Encryption Algorithm used is the default ("AES-128"). Append the string "CEA=" to the buffer. Append the string for the Client IP Encryption Algorithm to be used.

   B. If the Client IP Key Identifier is needed, perform this step. Append an "&" character to the buffer. Append the string "CKI=". Append the Client IP key identifier (e.g. "56128239") needed by the entity to locate the shared key for decrypting the Client IP.

   C. Append an "&" character. Append the string "CIP=".

   D. Convert the client’s IP address in CIDR notation (dotted decimal format for IPv4 or canonical text representation for IPv6 [RFC5952]) to a string and encrypt it using AES-128 (in ECB mode) or another algorithm if specified by the CEA Information Element.

   E. Convert the encrypted Client IP to its equivalent hexadecimal format.

   F. Append the value computed in the previous step to the buffer.

5. If a Key ID Information Element is needed, perform this step. Append an "&" character to the buffer. Append the string "KID=" in case a string-based Key ID is used, or "KID_NUM=" in case a numerical Key ID is used. Append the key identifier (e.g. "example:keys:123" or "56128239") needed by the entity to locate the shared key for validating the URI signature.

6. If a symmetric shared key is used, perform this step. However, skip this step when the hash function for the HMAC uses the default value ("SHA-256"). Append an "&" character to the buffer. Append the string "HF=". Append the string for the new type of hash function to be used.

7. If asymmetric public/private keys are used, perform this step. However, skip this step if the digital signature algorithm uses the default value ("ECDSA"). Append an "&" character to the buffer. Append the string "DSA=". Append the string for the digital signature function.

8. Append an "&" character. Append the string "UPC=".

9. Append the value of the URI Pattern Container in the form of a string to the buffer. Note: the value of the URI Pattern Container element should be appropriate given the file and folder structure of the HTTP Adaptive Streaming content in question. As an example, if the URL to the manifest file is 'http://example.com/folder/content-83112371/manifest.xml', a suitable URI Pattern might be '*://*/folder/content-83112371/quality_*/segment????.mp4'. If the manifest file and segments are stored in different paths, it is possible to concatenate multiple URI Patterns in a single URI Pattern Container Information Element, such as '*://*/folder/content-83112371/manifest/*.xml;*://*/folder/content-83112371/quality_*/segment????.mp4'.

10. Append an "&" character. Append the string "STT=". Append the value of the Signed Token Transport IE as a string to the buffer. In case the Cookie-based mechanism described in Section 5 is used, this value is set to 1.

11. If symmetric keys are used, perform this step.

    A. Obtain the shared key to be used for signing the Signed Token

    B. Append the string "MD=". The buffer now contains the complete section of the Signed Token that is protected (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&MD=").

    C. Compute the message digest using the HMAC algorithm and the default SHA-256 hash function, or another hash function if specified by the HF Information Element, with the shared key and message as the two inputs to the hash function.

    D. Convert the message digest to its equivalent hexadecimal format.

    E. Append the string for the message digest (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/

content-83112371/quality_*/segment????.mp4&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").

12. If asymmetric public/private keys are used, perform this step.

    A. Obtain the private key to be used for signing the Signed Token.

    B. Append the string "DS=". The message now contains the complete section of the Signed Token that is protected. (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&DS=").

    C. Compute the message digest using SHA-1 (without a key) for the message. Note: The digital signature generated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message. This is done to reduce the length of the digital signature and the resulting Signed URI. Since SHA-1 is not used for cryptographic purposes here, the security concerns around SHA-1 do not apply.

    D. Compute the digital signature, using the EC-DSA algorithm by default or another algorithm if specified by the DSA Information Element, with the private EC key and message digest (obtained in previous step) as inputs.

    E. Convert the digital signature to its equivalent hexadecimal format.

    F. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the ’r’ and ’s’ parameters, delimited by ’:’ (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E")

13. The buffer now contains the complete Signed Token. The Signed Token is packaged and transported to the UA as defined in Section 5

5. Communicating a Signed Token

The following steps describe the mechanism used for transporting a Signed Token to a UA as part of an HTTP 2xx Successful message or an HTTP 3xx Redirection message. The steps below assume the value of the Signed Token Transport (STT) Information Element has been set to 1. Other values of STT are out of scope of this document.

5.1. Communicating the Signed Token via Cookie

The Signed Token is communication to the UA via HTTP Cookies. By using standard HTTP Cookies, current UAs do not need to be adapted for them to work with the Signed Token Chain mechanism described in this document.

1. Encode the Signed Token by applying Base-64 Data Encoding [RFC4648] on the value of the Signed Token.

2. Add a 'URISigningPackage' cookie to the HTTP 2xx Successful 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. Set the value of the Cookie to the Base-64 encoded Signed Token obtained in the previous step.

5.2. Support for cross-domain redirection

For security purposes, use of cross-domain cookies is not supported in some application environments. Because of this, 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. To allow for this scenario, the steps below SHOULD be used in such cases instead of the process defined in Section 5.1. They MUST NOT be used in combination with HTTP 2xx Successful messages. Note that the process described below only works in cases where both the manifest file and segments constituting the HTTP Adaptive Streaming 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.

1. Copy the entire Original URI into a buffer to hold the message.

2. Check if the Original URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.

3. Append the parameter name used to indicate the URI Signing Package Attribute, as communicated via the CDNI Metadata interface or set by configuration, followed by an "=". If none

is communicated by the CDNI Metadata interface or set by configuration, it defaults to "URISigningPackage". For example, if the CDNI Metadata interface specifies "SIG", append the string "SIG=" to the message.

4. Encode the Signed Token by applying Base-64 Data Encoding [RFC4648] on the value of the Signed Token.

5. Append the URI Signing token to the message (e.g. "http://example.com/folder/content-83112371/manifest.xml?URISigningPackage=VkVSPTImYW1wO0VUPTEyMDk0MjI5NzYmYW1wO0VUUz0xNSZhbXA7Q0lQPTE5Mi4wLjIuMSZhbXA7VVBDPSo6Ly8qL2ZvbGRlci9jb250ZW50LTgzMTEyMzcxL3F1YWxpdHlfKi9zZWdtZW50Pz8/Py5tcDQmYW1wO0tJRD1leGFtcGxlOmtleXM6MTIzJmFtcDtNRD0xZWNiMTQ0NmE2NDMxMzUyYWFiMGZiNmUwZGNhMzBlMzAzNTY1OTNhOTdhY2I5NzIyMDIxMjBkYzQ4MmJkZGFm").

6. Place the message in the Location header of the HTTP 3xx Redirection message returned to the UA.

6. Receiving a Signed Token

The following steps describe the mechanism used for receiving a Signed Token as part of an HTTP GET request. The steps below assume the value of the Signed Token Transport (STT) Information Element has been set to 1 and the mechanism described in Section 5 has been used to send the Signed Token to the UA. Other values of STT are out of scope of this document.

1. Check if the query string component of the received URI contains the 'URISigningPackage' attribute. If there are multiple instances of this attribute, the first one is used and the remaining ones are ignored. This ensures that the Signed Token can be validated despite a client appending another instance of the URI Signing Package attribute. If the 'URISigningPackage' attribute is present, the Signed Token was sent to the UA as part of a cross-domain HTTP 3xx Redirection message. Extract the value of the 'URISigningPackage' attribute and use that as the URI Signing Package for the Signed Token validation procedure defined in Section 7.

2. If the query string component of the received URI does not contain the 'URISigningPackage' attribute, check if the HTTP request contains a 'URISigningPackage' cookie and use that as the URI Signing Package for the Signed Token validation procedure defined in Section 7.

3. If the request does not contain the 'URISigningPackage' query string attribute, does not contain a URISigningPackage cookie,

and the server doesn't support values of STT other than '1', the request is denied.

7. Validating a Signed Token

The process of validating a Signed Token can be divided into four sets of steps: 1) Decode URI Signing Package and extract the URI Signing Information Elements, 2) Validate the signature of the Signed Token to ensure its integrity, 3) Validate the Enforcement Information Elements to ensure proper enforcement of the distribution policy, and 4) Generate a subsequent Signed Token and communicate it to the UA.

In the algorithm below, the integrity of the Signed Token is confirmed before distribution policy enforcement because validation procedure would detect the right event when the URI is tampered with. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

7.1. Decode URI Signing Package and Information Element Extraction

Decode the URI Signing Package obtained as defined in Section 6 and extract the URI Signing Information Elements. Note that some steps are to be skipped if the corresponding URI Signing Information Elements are not embedded in the Signed Token.

1. Decode the URI Signing Package using Base-64 Data Encoding [RFC4648] to obtain all the URI Signing Information Elements in the form of a concatenated string (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").

2. Extract the value from "VER" if the Information Element exists. Determine the version of the URI Signing algorithm used to process the Signed URI or Signed Token. If the CDNI Metadata interface is used, check to see if the used version of the URI Signing algorithm is among the allowed set of URI Signing versions specified by the metadata. If this is not the case, the request is denied. If the Information Element is not in the Information Elements string, then it is assumed to be set to '1'. In that case, the Signed URI validation mechanism specified in [I-D.ietf-cdni-uri-signing] should be followed instead of the Signed Token mechanism defined in this document.

3. If the value of the "VER" Information Element is set to a value unequal to '2', the URI Signing Package refers to a different

version of URI Signing and the algorithm specified in this section shouldn't be used.

4. Extract the value from "MD" if the Information Element exists. The existence of this Information Element indicates a symmetric key is used.

5. Extract the value from "DS" if the Information Element exists. The existence of this Information Element indicates an asymmetric key is used.

6. If neither the "MD" or "DS" attribute exists, then no URI Signature exists and the request is denied. If both the "MD" and the "DS" Information Elements are present, the Signed Token is considered to be malformed and the request is denied.

7. Extract the value from "UPC". If the Information Element doesn't exist, the Signed Token is considered to be malformed and the request is denied.

8. Extract the value from "CIP" if the Information Element exists. The existence of this Information Element indicates content delivery is enforced based on client IP address.

9. Extract the value from "ET" if the Information Element exists. The existence of this Information Element indicates content delivery is enforced based on time.

10. Extract the value from the "KID" or "KID_NUM" Information Element if they exist. The existence of either of these Information Elements indicates a key can be referenced. If both the "KID" and the "KID_NUM" Information Elements are present, the Signed Token is considered to be malformed and the request is denied.

11. Extract the value from the "HF" Information Element if it exists. The existence of this Information Element indicates a different hash function than the default.

12. Extract the value from the "DSA" Information Element if it exists. The existence of this Information Element indicates a different digital signature algorithm than the default.

13. Extract the value from the "CEA" Information Element if it exists. The existence of this Information Element indicates a different Client IP Encryption Algorithm than the default.

14. Extract the value from the "CKI" Information Element if it exists. The existence of this Information Element indicates a key can be referenced using which the Client IP was encrypted.

15. Extract the value from "STT". If the Information Element doesn't exist, the Signed Token is considered to be malformed and the request is denied. If STT is equal to '1', the Cookie-based transport mechanism defined in Section 5 is used for returning the new Signed Token. If STT is unequal to '1', the new Signed Token is transported via a method that is not defined by this document.

16. Extract the value from "ETS" if the Information Element exists. This Information Element indicates the validity time of the next Signed Token in the chain.

7.2. Signature Validation

Validate the URI Signature of the Signed Token.

1. Copy the decoded URI Signing Package into a new buffer to hold the message for performing the operations below.

2. Based on the presenence of either the MD or DS Information Element in the decoded Signed Token, validate the message digest or digital signature for symmetric or asymmetric keys, respectively.

3.

   A. For MD, an HMAC algorithm is used.

      a. If either the "KID" or "KID_NUM" Information Element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or "KID_NUM" Information Element is present in the received URI Signing Package, obtain the shared key via CDNI metadata or configuration.

      b. If the "HF" Information Element exists, validate that the hash function is in the allowable "HF" set as listed in the CDNI metadata or configuration. The request is denied when the hash function is not allowed. If the "HF" Information Element is not in the received URI Signing Package, the default hash function is SHA-256.

c. Convert the extracted value of the MD element to binary format. This will be used to compare with the computed value later.

d. Remove the value part of the "MD" Information Element (but not the '=' character) from the buffer. The message is ready for validation of the message digest (e.g. "VER =2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/ content-83112371/ quality_*/segment????.mp4&KID=example:keys:123&MD=").

e. Compute the message digest using the HMAC algorithm with the shared key and message as the two inputs to the hash function.

f. Compare the result with the received message digest to validate the Signed Token. If there is no match, the request is denied.

B. For DS, a digital signature function is used.

a. If either the "KID" or "KID_NUM" Information Element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or "KID_NUM" Information Element is present in the received URI Signing Package, obtain the public key via CDNI metadata or configuration.

b. If the "DSA" Information Element exists, validate that the digital signature algorithm is in the allowable "DSA" set as listed in the CDNI metadata or configuration. The request is denied when the DSA is not allowed. If the "DSA" Information Element is not in the received URI Signing Package, the default DSA is EC-DSA.

c. Convert the extracted value of the DS element to binary format. This will be used for verification later.

d. Remove the value part of the "DS" Information Element (but not the '=' character) from the message. The message is ready for validation of the digital signature (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*:// */folder/content-83112371/ quality_*/segment????.mp4&KID=http://example.com/public/ keys/123&DS=").

e. Compute the message digest using SHA-1 (without a key) for the message.

f. Verify the digital signature using the digital signature function (e.g. EC-DSA) with the public key, received digital signature, and message digest (obtained in previous step) as inputs. This validates the Signed Token. If signature is determined to be invalid, the request is denied.

7.3. Distribution Policy Enforcement

Note that some of the steps below are to be skipped if the corresponding URI Signing Information Elements are not in the received URI Signing Package. The absence of a given Enforcement Information Element indicates enforcement of its purpose is not necessary in the CSP's distribution policy. The exception is the URI Pattern Container Information Element, which is mandatory for Signed Tokens.

1. If the "CIP" Information Element does not exist, this step can be skipped.

   A. Obtain the key for decrypting the Client IP, as indicated by the Client IP Key Index Information Element or set via configuration.

   B. Decrypt the encrypted Client IP address communicated through the Client IP Information Element using AES-128, or the algorithm specified by the Client IP Encryption Algorithm Information Element.

   C. Verify, using CIDR matching, that the request came from an IP address within the range indicated by the decrypted Client IP Information Element. If the IP address is incorrect, the request is denied.

2. If the "ET" Information Element exists, validate that the request arrived before expiration time based on the Expiration Time Information Element. If the time expired, the request is denied.

3. Validate that the requested resource is in the allowed set by matching the received URI against each of the URI Patterns in the URI Pattern Container Information Element until a match is found. If there is no match, the request is denied.

7.4. Subsequent Signed Token Generation

The following steps describe how to generate a subsequent Signed Token in a Signed Token Chain. Note that the process for generating an initial Signed Token is described in Section 4 and the process below is used for generating all subsequent tokens after the initial one.

1. Create a new buffer for constructing the new Signed Token in the steps below.

2. Append the string "VER=2"

3. If the received URI Signing Package contains the "ET" Information Element, perform this step.

    A. Append an "&" character to the buffer. Append the string "ET=".

    B. If the received URI Signing Package contains the "ETS" Information Element, perform this step.

        1. Get the value of the "ETS" Information Element and convert it to an integer.

        2. Get the current time in seconds sinds epoch (as an integer) and add the value of the "ETS" Information Element as seconds.

        3. Convert the result to a string and append it to the buffer.

        4. Append the "&" character and the "ETS=" string to the buffer.

        5. Append the value of the "ETS" Information Element in the received URI Signing Package to the buffer.

    C. If the received Signed Token does not contain the "ETS" Information Element, perform this step. Get the value of the "ET" Information Element from the received URI Signing Package and append it to the buffer.

4. If the received URI Signing Package contains the "CIP" Information Element, perform this step.

A. Append an "&" character to the buffer. Append the string "CIP=". Append the value of the "CIP" Information Element in the received URI Signing Package.

B. If the received URI Signing Package contains the "CEA" Information Element, perform this step. Append an "&" character to the buffer. Append the string "CEA=" to the buffer. Append the value of the "CEA" Information Element in the received URI Signing Package.

C. If the received URI Signing Package contains the "CKI" Information Element, perform this step. Append an "&" character to the buffer. Append the string "CKI=". Append the value of the "CKI" Information Element in the received URI Signing Package.

5. If a Key ID Information Element is needed, perform this step. Append an "&" character to the buffer. Append the string "KID=" in case a string-based Key ID is used, or "KID_NUM=" in case a numerical Key ID is used. Append the key identifier (e.g. "example:keys:123" or "56128239") needed by the entity to locate the shared key for validating the URI signature.

6. If symmetric keys are used, perform this step. If the hash function for the HMAC uses the default value ("SHA-256"), this step can be skipped. Append an "&" character to the buffer. Append the string "HF=". Append the string for the new type of hash function to be used.

7. If asymmetric keys are used, perform this step. If the digital signature algorithm uses the default value ("EC-DSA"), this step can be skipped. Append an "&" character to the buffer. Append the string "DSA=". Append the string for the digital signature function.

8. Append an "&" character to the buffer. Append the string "UPC=". Append the value of the "UPC" Information Element in the received URI Signing Package.

9. Append an "&" character to the buffer. Append the string "STT=". Append the value of the "STT" Information Element in the received URI Signing Package.

10. Depending on the type of key used to sign the received Signed Token, compute the message digest or digital signature for symmetric key or asymmetric keys, respectively.

A. If asymmetric keys are used, this step can be skipped.

1. Obtain the shared key to be used for signing the Signed Token.

2. Append an "&" character to the buffer. Append the string "MD=". The message now contains the complete set of URI Signing Information Elements over which the URI Signature is computed (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&MD=").

3. Compute the message digest using the HMAC algorithm and the default SHA-256 hash function, or another hash function if specified by the HF Information Element, with the shared key and message as the two inputs to the hash function.

4. Convert the message digest to its equivalent hexadecimal format.

5. Append the string for the message digest to the buffer (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&MD=d6117d7db8a68bd59f6e7e3343484831acd8f23bbaa7f44b285a2f3bb6f02cfd").

B. If symmetric keys are used, this step can be skipped.

1. Obtain the private key to be used for signing the Signed Token.

2. Append the string "DS=". The message now contains the complete section of the Signed Token that is protected. (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&DS=").

3. Compute the message digest using SHA-1 (without a key) for the message. Note: The digital signature generated in the next step is calculated over the SHA-1 message digest, instead of over the cleartype message. This is done to reduce the length of the digital signature and the resulting Signed URI. Since SHA-1 is not used for cryptographic purposes here, the security concerns around SHA-1 do not apply.

4. Compute the digital signature, using the EC-DSA algorithm by default or another algorithm if specified by the DSA Information Element, with the private EC key

and message digest (obtained in previous step) as inputs.

5. Convert the digital signature to its equivalent hexadecimal format.

6. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the 'r' and 's' parameters, delimited by ':' (e.g. "VER=2&ET=1209422976&ETS=15&CIP=192.0.2.1&UPC=*://*/folder/content-83112371/quality_*/segment????.mp4&KID=example:keys:123&DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E")

11. The buffer now contains the complete Signed Token. The Signed Token is packaged and transported to the UA as defined in Section 5

8. IANA Considerations

[Editor's note: TO DO]

9. References

9.1. Normative References

[I-D.ietf-cdni-uri-signing]
Leung, K., Faucheur, F., Brandenburg, R., Downey, B., and M. Fisher, "URI Signing for CDN Interconnection (CDNI)", draft-ietf-cdni-uri-signing-07 (work in progress), April 2016.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://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, <http://www.rfc-editor.org/info/rfc6707>.

9.2. Informative References

[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, <http://www.rfc-editor.org/info/rfc6983>.

Author's Address

Ray van Brandenburg
TNO
Anna van Buerenplein 1
Den Haag 2595DC
the Netherlands

Phone: +31 88 866 7000
Email: ray.vanbrandenburg@tno.nl

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

F. Fieau, Ed.
E. Stephan
Orange

July 8, 2016

Limited Use of Remote Keys for Interconnected CDNs
draft-cdni-fieau-lurk-https-delegation-00

Abstract

Sharing private keys amongst administrative entities raises numerous issues for end-users, intermediaries and content origins. The IETF envisions the standardization of protocols to limit the exchange of private keys (LURK). This document presents CDN Interconnection (CDNI) use cases based on the Limited Use of Remote Keys (LURK) protocol and architecture and identifies a set of requirements.

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 http://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 January 9, 2017.

Copyright Notice

Table of Contents

1. Introduction

CDNI requirements [RFC7337] and use cases [RFC6770] specify the requirements and use cases of HTTP content delivery between CDNs. The intent of this document is to pursue the work by proposing a solution for delegating content delivery over secure protocols like

HTTPS or DTLS. Conversely to HTTP delivery, HTTPS [RFC2818] allows content delivery between Content Service Provider (CSP) and End-users with integrity and confidentiality.

From the CDNI perspective, all parties - UAs, CSPs origin servers, and CDNs - are involved in the chain of trust and preserving this chain of trust in HTTPS raises concerns.

Indeed, regarding TLS certificates, CSPs and CDN providers are reluctant to share private keys mainly because of legal and security issues about private keys. Therefore the CDNI working group must specify a solution that avoids the exchange of private keys between CDNs.

This document leverages the work of the LURK initiative [LURK_Mailing_List] and discusses the introduction of a Limited Use of Remote Keys (LURK) solution in the CDNI architecture. It presents 2 use cases which complement those described in [I-D.mglt-lurk-tls-use-cases] and identifies a set of requirements for CDNI. Finally it discusses different options and identifies a set of open issues.

The proposed use cases are based on DNS redirection. The user agent is redirected to a dCDN to establish a TLS session to get HTTPS content. Additional use cases are expected in future versions of the draft.

This version focus on the delivery over HTTPS only.

2. Terminology

This document uses terminology from CDNI framework documents such as CDNI requirements [RFC7337] and CDNI interface specifications documents - CDNI metadata interface [I-D.ietf-cdni-metadata], CDNI Control interface [I-D.ietf-cdni-control-triggers] and Logging interface [I-D.ietf-cdni-logging].

3. CDNI architecture using LURK

This document introduces a Key Server in the CDNI architecture. The general LURK architecture is described in the Session Key interface draft [I-D.cairns-tls-session-key-interface].

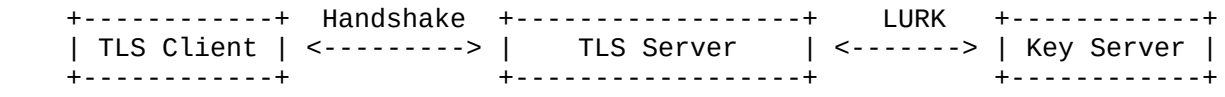


Figure 1: General Key Server Architecture

In CDNI, a LURK interface allows private keys to remain under the authority of its owner - typically the CSP or the uCDN - while delivering the content over HTTPS.

The LURK interface can be introduced at different locations in the CDNI architecture. The location of the LURK function depends on the use cases. Additionally, this architecture may support variants where the Key server is owned by a third party.

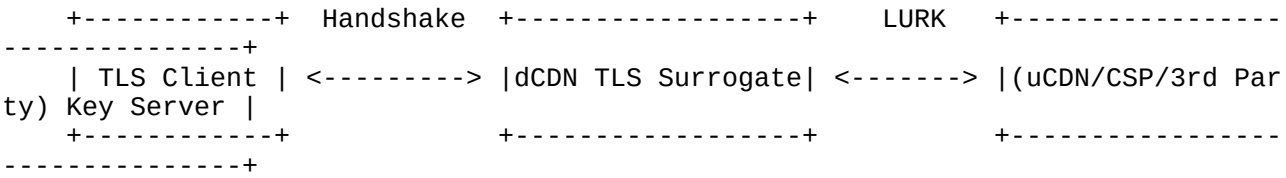


Figure 2: Key Server in CDNI Architecture

This document complements the LURK architecture of [I-D.mglt-lurk-tls-use-cases] with CDNI use cases. In those use cases, content delivery is decoupled from both content acquisition and signaling information needed to route and control the request.

Currently CDNI signaling exchanges occur over the Request Routing Redirection interface (information to route the request across CDNs), the Metadata interface (per content or group of content information about the acquisition and the delivery), the Logging interface (exchange of monitoring information about the delivery) and the Control interface (information for controlling the lifecycle of the aforementioned interfaces).

The LURK interface for CDNI adds two types of exchange: one for acquiring the certificate associated with the origin domain and the other for establishing delivery confidentiality and integrity.

This document presents two use cases of HTTPS delivery which rely on a LURK function to avoid exchanging private keys between CDNs:

A. uCDN Key server: the CSP has provided his certificate and private keys to the uCDN. the uCDN provides the LURK key server and interface.

B. CSP Key server: the CSP provides the LURK Key Server and interface.

The table below presents the use cases developed in the Section 3.

Use Case name	UA Redirection	uCDN redirection	CSP Cert delegation
UC A: uCDN KS	DNS CNAME	recursive	uCDN
UC B: CSP KS	DNS CNAME	recursive	No

Figure 3: Use cases description table

The use cases' call flows are respectful of the CDNI redirection [I-D.ietf-cdni-redirection] for the description of redirection methods in CDNI.

They share the same steps.

The figure below illustrates the framework use cases where the surrogate doesn't handle the CSP private keys and where the surrogate remotely accesses materials to sign and encrypt information exchanged over the TLS connection.

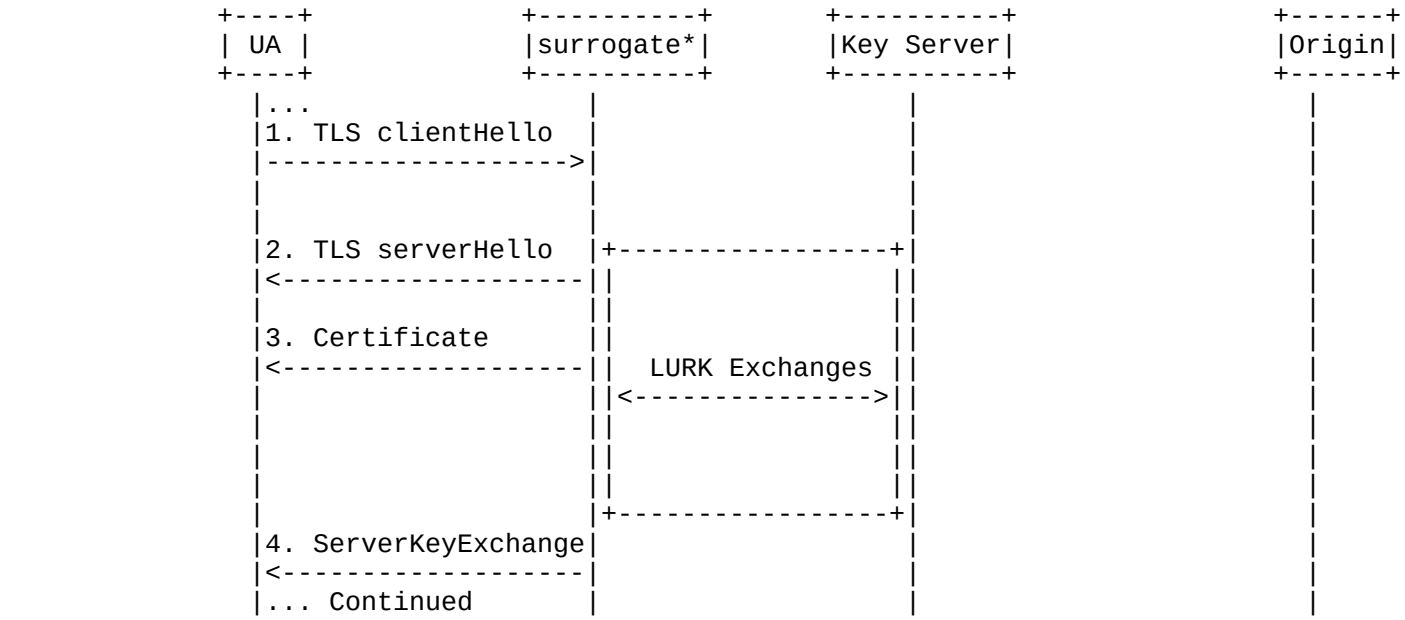


Figure 4: LURK exchanges in CDNI architecture

4. LURK use cases for CDNI

Two use cases are considered in this document to implement LURK in CDNI:

Use Case A. uCDN Key Server

Use Case B. CSP Key Server

4.1. Use case A: uCDN Key Server

In this use case, the dCDN has a LURK interface to a Key Server hosted at the uCDN side. It may be typically a case of CDNI regional delivery delegation.

This following example is based on ECDHE cypher suite. The UA is first redirected from the uCDN to a dCDN using a recursive DNS redirection. Then the UA initiates a TLS connection with the dCDN to get his content. Since the dCDN does not store the private keys for the requested certificate, it queries the uCDN Key Server (KS) to get the credential to establish the TLS session with the User Agent. Finally the dCDN can deliver the content in HTTPS to the UA using the CSP certificate.

The UA sends an HTTPS request to the CSP to get a content.

a. to d. As seen on figure 5, once the DNS resolution is over, i.e., UA was able to resolve dCDN surrogate IP@ steps [a.] to [d.], the user agent connects to the dCDN surrogate. Note that DNS cache is not shown on the figure.

1. The UA sends a ClientHello, as defined in the TLS protocol [RFC5246]. The SNI field of the ClientHello includes the CSP domain name.

2. The surrogate receives the ClientHello from the UA, and sends a ServerHello to the UA.

3. The surrogate parses the SNI field, and determines the Key Server interface of the CSP domain name. The surrogate uses this piece of information to determine the certificate of the delivery. The surrogate sends the public certificate to the UA. The UA validates the certificate.

4. The surrogate determines the ECDHE parameters, and requests the Key Server to sign these parameters with the CSP private key. The request includes the domain name received by the surrogate in the SNI field of the ClientHello.

5. The Key Server returns the necessary credentials to the dCDN surrogate over a secure tunnel.

6. and 7. the UA and the surrogate exchange public DH parameters and compute session keys.

8. The UA and the surrogate establish a secure connection. The UA issues its request for content over HTTPS.

The surrogate then processes the original request.

Below is an example of the handshake establishment:

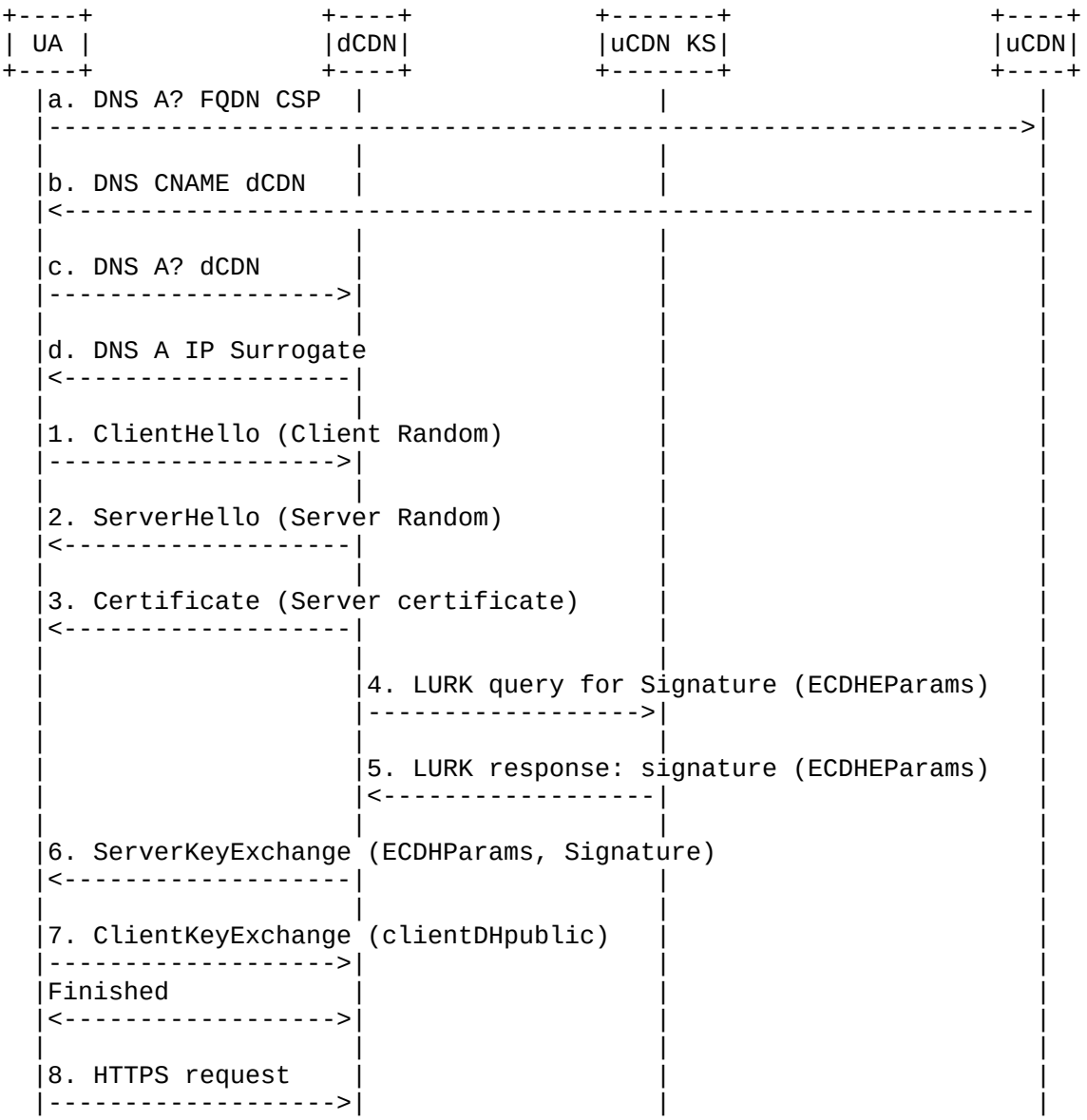


Figure 5: Key Server hosted by uCDN

4.2. Use case B: CSP Key Server

This section describes a use case in CDNI where the CSP provides the Key Server (KS).

In this example, the CSP delegates the HTTPS content delivery to an uCDN that in turn delegates the HTTPS delivery to a dCDN. For obvious legal or security reasons, the CSP does not want his private keys to be stored on all CDNs involved in the interconnection. In that case, the dCDN relies on credentials received from a CSP Key Server (KS) to deliver HTTPS content.

The dCDN cannot here request the KS directly. Instead, the dCDN LURK request will be relayed by the uCDN to the Key Server. Prior to that, the uCDN has to check whether the received LURK request is valid, e.g., complies with Content Distribution policies [I-D.ietf-cdni-metadata].

In the following example, the CSP stores his certificates and private keys on a Key Server that is able to compute and provide credentials for TLS establishment.

1. The UA sends a ClientHello to the dCDN’s surrogate, as defined in the TLS protocol [RFC5246]. The SNI field of the ClientHello includes the CSP domain name.

2. The dCDN’s surrogate receives the ClientHello from the UA, and sends a ServerHello to the UA..

3. The surrogate parses the SNI field, and determines the Key Server interface of the CSP domain name and determine the certificate of the delivery. The surrogate sends the public certificate to the UA. The UA checks the certificate signature with the public key.

4. The dCDN’s surrogate requests the uCDN to sign parameters with the private key.

5. The uCDN parses the SNI field, and determines the Key Server interface of the CSP domain name. The uCDN relaies the LURK request received from the dCDN’s surrogate, to the Key Server to sign parameters with the private key.

6. The Key Server returns the necessary credentials to the uCDN surrogate.

7. The uCDN returns the necessary credentials to the dCDN’s surrogate.

8. and 9. the UA and the surrogate exchange public DH parameters and compute session keys.

10. The UA and the surrogate establish a secure connection. The UA issues its request for content over HTTPS.

The surrogate then processes the original request.

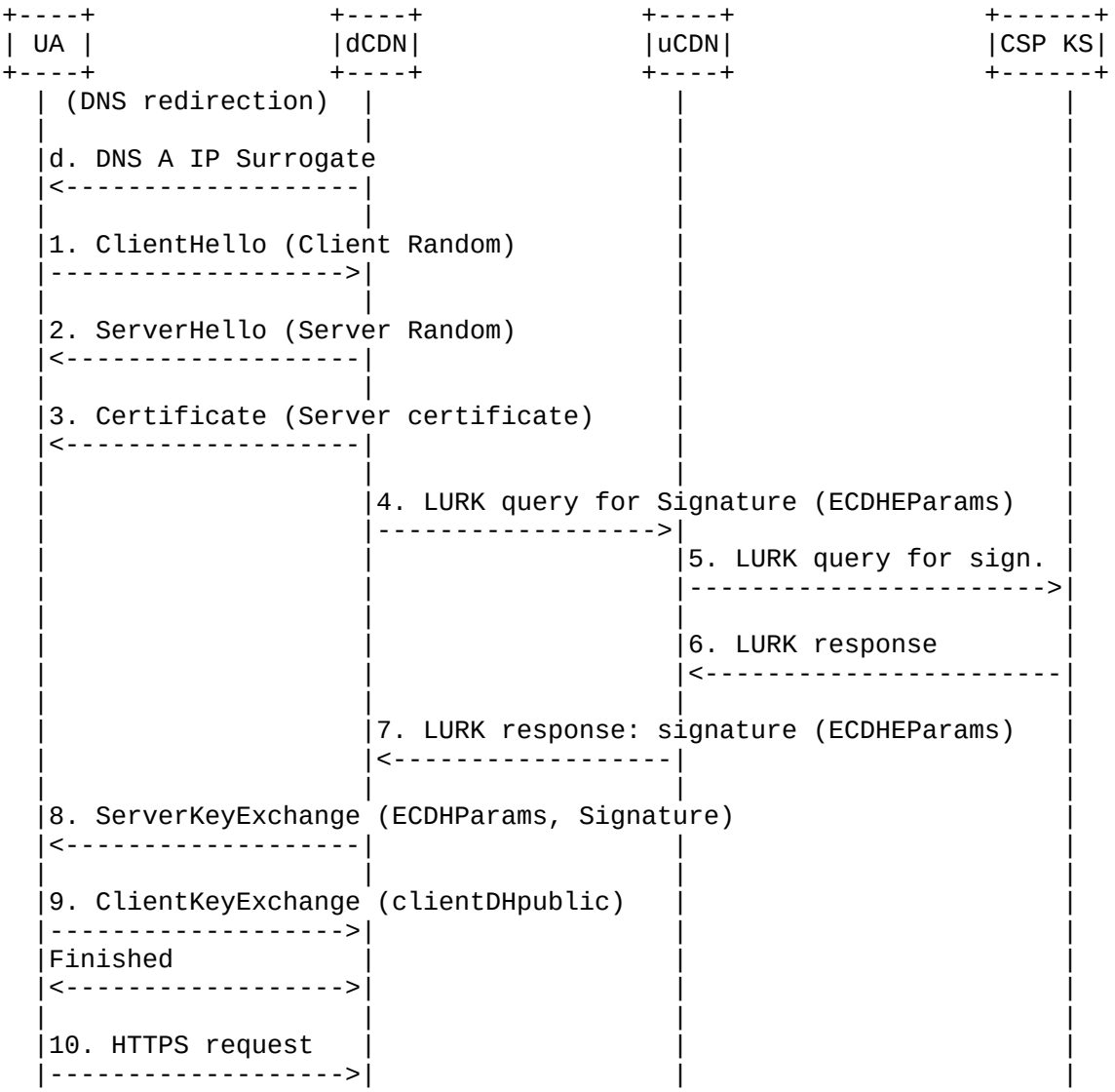


Figure 6: Cascaded delegation with LURK

4.3. Other use cases

Other use cases will be described in a further version of this draft.

5. Requirements

This section is a first attempt to identify the requirements introduced by the delivery of HTTPS content. Some of the requirements are new, whereas others may update existing CDNI requirements [RFC7337].

5.1. General

LURK-GEN-1: The specification should not require any change in User Agent. This is already stated in [RFC7337]/GEN-2

Discussion: Despite this is obvious, it is important to notice that recent limitation in TLS (version, cyphers...) or HTTP (cyphers) documents may constrain User Agent.

LURK-GEN-2: The user agent should be able to see that the requested content is delivered by the CDN using the CSP domain. Delivery domain name SHOULD be the same as the CSP portal domain name (or a sub-domain request name)

LURK-GEN-3: The Certificate delivered by the dCDN and CSP must match the domain of the URL [RFC2818]. As an example, it means that no certificate error occurs on the UA when the dCDN has redirected the UA a dCDN.

LURK-GEN-4: The dCDN’s surrogates which implements LURK interface should support the delivery of content of the protocol HTTPS.

LURK-GEN-5: The dCDNs must be able to present the CSP certificate and credentials to the user agent when establishing a TLS session

5.2. CDNI Control Interface requirements

LURK-CI-1: The CDNI Control Interface may allow the bootstrapping of a Key Server interface. For example this may include:

- discovery the Key Server interface endpoint.

- credentials for Key Server and CDN mutual authentication.

- TLS version, Certificate types, TLS crypto suites.

Proposal: add this requirement to the section CDNI Control Interface of [RFC7337].

5.3. CDNI Footprint and Capabilities Advertisement interface requirements

LURK-FC-1: The CDNI Footprint and Capabilities Advertisement interface should allow the downstream CDN to communicate to the upstream CDN about its capabilities regarding the support of client side of the Key Server interface.

5.4. CDNI Logging Interface requirements

This section identifies additional requirements related to the CDNI Logging interface (LI).

TLS-LI-1: the CDNI Logging interface should allow a dCDN to report to the uCDN logging for deliveries which fail during the establishment of the secure connection, e.g., ability to report certificate validation error.

TLS-LI-2: The CDNI Logging interface should allow dCDN to report to uCDN logging incomplete deliveries due to encryption errors. Discussion: this requirement is mostly a subcase of LI-2 about incomplete deliveries.

LURK-LI-3: the CDNI Logging interface should allow a dCDN to report to the uCDN secure connections failures when using LURK interface.

As an example, the UA can’t establish a secure connection to a dCDN because either, the credentials provided by a Key Server are invalid, or because the Key Server has refused to provide them to the dCDN.

5.5. Key Server Interface requirements

LURK-KS-1: A Key Server interface must not allow the exchange private keys. This is the centrality of the LURK interface.

LURK-KS-2: The Key Server and the requesting CDN must authenticate each other, according to the information provided by the CDNI Control interface.

LURK-KS-3: The dCDN Key Server interface must be able to send a LURK request to a Key Server.

As an example (UC A, step 4), the dCDN surrogate determines ECDHE parameters (...), and requests the Key Server to sign these

parameters with the CSP private key. The request includes the domain name received by the surrogate in the SNI field of the ClientHello.

6. Discussions

6.1. HTTPS and TLS

HTTPS contents are delivered with the dCDN credentials. The introduction of a Key Server in the CDNI architecture allows the HTTPS content to be delivered with the CSP origin server certificate. It conforms to the end-to-end HTTPS framework [RFC7230].

6.2. Cyphersuites in CDNI

CDNI WG and LURK WG should use a common set of cyphersuites, or CDNI WG should specify or points to the set of suites to support.

TLS and HTTP2 recommend different set of cypher suites. CDNI WG should clarify which one should be supported in the case of a HTTPS delivery based on LURK credentials: HTTP2 Cipher Suite, refer to appendix A of [RFC7540] and "backwards-compatibility-security-restrictions" in [I-D.draft-ietf-tls-tls13"].

6.3. PFS

Should the delivery be PFS proof?

6.4. TLS version

Which version of TLS should be supported by LURK: TLS/LTS, TLS1.2, TLS1.3?

6.5. Scalability

For each TLS session initialization on the dCDNs, the dCDNs need to request the KS to get the necessary credentials. To be scalable, the KS may need to be replicated.

6.6. Certificates and security

At least one private key per CSP is stored on the KS. Therefore the use of a KS avoids the complexity of duplicating private keys on uncontrolled servers. This also allows the uCDN to maintain a single domain name for the CDN interconnection.

6.7. Revocation

In the case of an HTTPS delegation revocation, a dCDN has no longer the delegation right to deliver a content for a given CSP. The uCDN would then deny access to CSP certificates and credentials derived from private keys, and therefore the dCDN would not be able to establish the TLS session without triggering a warning on UA Side.

6.8. Certificate provisioning

The dCDN may have to retrieve at least once the CSP public certificate related to the targeted domain name. The certificates may be cached on the dCDN for a given duration. .

Is certificate provisioning in the scope of CDNI as it seems implementation dependant? Which interface provides the certificates to the uCDN (Control, Metadata, LURK, ..)?

6.9. CSP side

With regard to the use case B, the CSP may provide a Key server. As a consequence the requirement [RFC7337]/GEN-3 should be updated accordingly.

6.10. Acquisition

Usage of the LURK interface when acquiring content: when the dCDN acquires content directly from the origin server, would it have to request first the KS to get the uCDN credentials ?

6.11. CDNI Control Interface vs CDNI Metadata Interface

What should be carried by the CI or the MI ?

7. Open issues

7.1. TLS session resuming

Not yet addressed in this document, contributors are welcome.

7.2. Stack Evolution

Contents might be delivered over other protocols than TCP and than HTTP/1.1 in a close future. The CDNI WG must discuss the support of HTTPS delivery over TLS/LTS, TLSv3, DTLS, QUIC and HTTP2.

8. IANA Considerations

This document has no IANA considerations.

9. Security Considerations

The entire document is about security.

10. Acknowledgments

Many thanks to Benoit Gaussen who contributed to this draft.

11. References

11.1. Normative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

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

[RFC6770] Bertrand, G., Ed., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, DOI 10.17487/RFC6770, November 2012, <http://www.rfc-editor.org/info/rfc6770>.

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

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

11.2. Informative References

[I-D.cairns-tls-session-key-interface]
Cairns, K., Mattsson, J., Skog, R., and D. Migault, "Session Key Interface (SKI) for TLS and DTLS", draft-cairns-tls-session-key-interface-01 (work in progress), October 2015.

[I-D.erb-lurk-rsalg]
Erb, S. and R. Salz, "A PFS-preserving protocol for LURK", draft-erb-lurk-rsalg-01 (work in progress), May 2016.

[I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-15 (work in progress), May 2016.

[I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-27 (work in progress), June 2016.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-18 (work in progress), June 2016.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-18 (work in progress), April 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

[I-D.mglt-lurk-tls-use-cases]
Migault, D., Ma, K., Salz, R., Mishra, S., and O. Dios, "LURK TLS/DTLS Use Cases", draft-mglt-lurk-tls-use-cases-02 (work in progress), June 2016.

[LURK_Mailing_List]
"LURK Mailing List", <https://mailarchive.ietf.org/arch/search/?email_list=lurk>.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

B. Niven-Jenkins
R. Murray
Velocix (Alcatel-Lucent)
M. Caulfield
Cisco Systems
K. Ma
Ericsson

July 8, 2016

CDN Interconnection Metadata
draft-ietf-cdni-metadata-19

Abstract

The Content Delivery Networks Interconnection (CDNI) metadata interface enables interconnected Content Delivery Networks (CDNs) to exchange content distribution metadata in order to enable content acquisition and delivery. The CDNI metadata associated with a piece of content provides a downstream CDN with sufficient information for the downstream CDN to service content requests on behalf of an upstream CDN. This document describes both a base set of CDNI metadata and the protocol for exchanging that metadata.

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 http://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 January 9, 2017.

Table of Contents

1. Introduction

Content Delivery Networks Interconnection (CDNI) [RFC6707] enables a downstream Content Delivery Network (dCDN) to service content requests on behalf of an upstream CDN (uCDN).

The CDNI metadata interface is discussed in [RFC7336] along with four other interfaces that can be used to compose a CDNI solution (CDNI Control interface, CDNI Request Routing Redirection interface, CDNI Footprint & Capabilities Advertisement interface and CDNI Logging interface). [RFC7336] describes each interface and the relationships between them. The requirements for the CDNI metadata interface are specified in [RFC7337].

The CDNI metadata associated with a piece of content (or with a set of content) provides a dCDN with sufficient information for servicing content requests on behalf of an uCDN, in accordance with the policies defined by the uCDN.

This document defines the CDNI metadata interface which enables a dCDN to obtain CDNI metadata from an uCDN so that the dCDN can properly process and respond to:

- Redirection requests received over the CDNI Request Routing Redirection interface [I-D.ietf-cdni-redirection].
- Content requests received directly from User Agents.

Specifically, this document specifies:

- A data structure for mapping content requests and redirection requests to CDNI metadata objects (Section 3 and Section 4.1).
- An initial set of CDNI Generic metadata objects (Section 4.2).
- A HTTP web service for the transfer of CDNI metadata (Section 6).

1.1. Terminology

This document reuses the terminology defined in [RFC6707].

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

- Object - a collection of properties.
- Property - a key and value pair where the key is a property name and the value is the property value or another object.

This document uses the phrase "[Object] A contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B".

1.2. Supported Metadata Capabilities

Only the metadata for a small set of initial capabilities is specified in this document. This set provides the minimum amount of metadata for basic CDN interoperability while still meeting the requirements set forth by [RFC7337].

The following high-level functionality can be configured via the CDNI metadata objects specified in Section 4:

- Acquisition Source: Metadata for allowing a dCDN to fetch content from a uCDN.
- Delivery Access Control: Metadata for restricting (or permitting) access to content based on any of the following factors:
  - Location
  - Time Window
  - Delivery Protocol
- Delivery Authorization: Metadata for authorizing dCDN user agent requests.
- Cache Control: Metadata for controlling cache behavior of the dCDN.

The metadata encoding described by this document is extensible in order to allow for future additions to this list.

The set of metadata specified in this document covers the initial capabilities above. It is only intended to support CDN interconnection for the delivery of content by a dCDN using HTTP/1.1 [RFC7230] and for a dCDN to be able to acquire content from a uCDN using either HTTP/1.1 or HTTP/1.1 over TLS [RFC2818].

Supporting CDN interconnection for the delivery of content using unencrypted HTTP/2 [RFC7540] (as well as for a dCDN to acquire content using unencrypted HTTP/2 or HTTP/2 over TLS) requires the registration of these protocol names in the CDNI Metadata Protocol Types registry Section 7.3.

Delivery of content using HTTP/1.1 over TLS or HTTP/2 over TLS SHOULD follow the guidelines set forth in [RFC7525]. Offline configuration of TLS parameters between CDNs is beyond the scope of this document.

2. Design Principles

The CDNI metadata interface was designed to achieve the following objectives:

1. Cacheability of CDNI metadata objects;

2. Deterministic mapping from redirection requests and content requests to CDNI metadata properties;

3. Support for DNS redirection as well as application-specific redirection (for example HTTP redirection);

4. Minimal duplication of CDNI metadata; and

5. Leveraging of existing protocols.

Cacheability can decrease the latency of acquiring metadata while maintaining its freshness, and therefore decrease the latency of serving content requests and redirection requests, without sacrificing accuracy. The CDNI metadata interface uses HTTP and its existing caching mechanisms to achieve CDNI metadata cacheability.

Deterministic mappings from content to metadata properties eliminates ambiguity and ensures that policies are applied consistently by all dCDNs.

Support for both HTTP and DNS redirection ensures that the CDNI metadata meets the same design principles for both HTTP and DNS based redirection schemes.

Minimal duplication of CDNI metadata improves storage efficiency in the CDNs.

Leveraging existing protocols avoids reinventing common mechanisms such as data structure encoding (by leveraging I-JSON [RFC7493]) and data transport (by leveraging HTTP [RFC7230]).

3. CDNI Metadata object model

The CDNI metadata object model describes a data structure for mapping redirection requests and content requests to metadata properties. Metadata properties describe how to acquire content from an uCDN, authorize access to content, and deliver content from a dCDN. The object model relies on the assumption that these metadata properties can be grouped based on the hostname of the content and subsequently on the resource path (URI) of the content. The object model associates a set of CDNI metadata properties with a Hostname to form a default set of metadata properties for content delivered on behalf of that Hostname. That default set of metadata properties can be overridden by properties that apply to specific paths within a URI.

Different Hostnames and URI paths will be associated with different sets of CDNI metadata properties in order to describe the required behaviour when a dCDN surrogate or request router is processing User Agent requests for content at that Hostname and URI path. As a result of this structure, significant commonality could exist between the CDNI metadata properties specified for different Hostnames, different URI paths within a Hostname and different URI paths on different Hostnames. For example the definition of which User Agent IP addresses should be grouped together into a single network or geographic location is likely to be common for a number of different Hostnames; although a uCDN is likely to have several different policies configured to express geo-blocking rules, it is likely that a single geo-blocking policy could be applied to multiple Hostnames delivered through the CDN.

In order to enable the CDNI metadata for a given Hostname and URI Path to be decomposed into reusable sets of CDNI metadata properties, the CDNI metadata interface splits the CDNI metadata into separate objects. Efficiency is improved by enabling a single CDNI metadata object (that is shared across Hostname and/or URI paths) to be retrieved and stored by a dCDN once, even if it is referenced by the CDNI metadata for multiple Hostnames and/or URI paths.

Important Note: Any CDNI metadata object A that contains another CDNI metadata object B can include a Link object specifying a URI that can be used to retrieve object B, instead of embedding object B within object A. The remainder of this document uses the phrase "[Object] A

contains [Object] B" for simplicity when a strictly accurate phrase would be "[Object] A contains or references (via a Link object) [Object] B". It is generally a deployment choice for the uCDN implementation to decide when to embed CDNI metadata objects and when to reference separate resources via Link objects.

Section 3.1 introduces a high level description of the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects, and describes the relationships between them.

Section 3.2 introduces a high level description of the CDNI GenericMetadata object which represents the level at which CDNI metadata override occurs between HostMetadata and PathMetadata objects.

Section 4 describes in detail the specific CDNI metadata objects and properties specified by this document which can be contained within a CDNI GenericMetadata object.

3.1. HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects

The relationships between the HostIndex, HostMatch, HostMetadata, PathMatch, PatternMatch and PathMetadata objects are described in Figure 1.

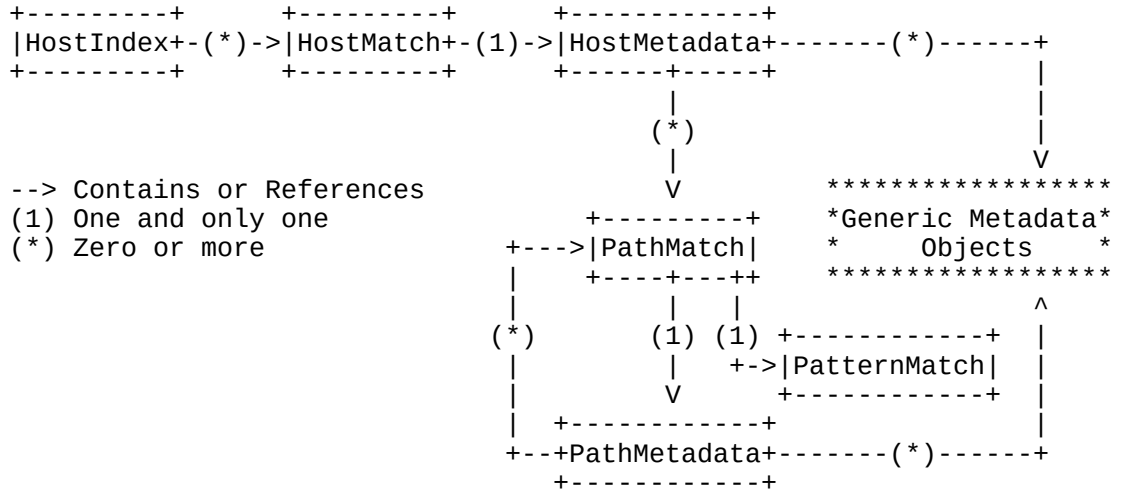


Figure 1: Relationships between CDNI Metadata Objects (Diagram Representation)

A HostIndex object (see Section 4.1.1) contains an array of HostMatch objects (see Section 4.1.2) that contain Hostnames (and/or IP addresses) for which content requests might be delegated to the dCDN. The HostIndex is the starting point for accessing the uCDN CDNI metadata data store. It enables the dCDN to deterministically discover which CDNI metadata objects it requires in order to deliver a given piece of content.

The HostIndex links Hostnames (and/or IP addresses) to HostMetadata objects (see Section 4.1.3) via HostMatch objects. A HostMatch object defines a Hostname (or IP address) to match against a requested host and contains a HostMetadata object.

HostMetadata objects contain the default GenericMetadata objects (see Section 4.1.7) required to serve content for that host. When looking up CDNI metadata, the dCDN looks up the requested Hostname (or IP address) against the HostMatch entries in the HostIndex, from there it can find HostMetadata which describes the default metadata properties for each host as well as PathMetadata objects (see Section 4.1.6), via PathMatch objects (see Section 4.1.4). PathMatch objects define patterns, contained inside PatternMatch objects (see Section 4.1.5), to match against the requested URI path. PatternMatch objects contain the pattern strings and flags that describe the URI path that a PathMatch applies to. PathMetadata objects contain the GenericMetadata objects that apply to content requests matching the defined URI path pattern. PathMetadata properties override properties previously defined in HostMetadata or less specific PathMatch paths. PathMetadata objects can contain additional PathMatch objects to recursively define more specific URI paths to which GenericMetadata properties might be applied.

A GenericMetadata object contains individual CDNI metadata objects which define the specific policies and attributes needed to properly deliver the associated content. For example, a GenericMetadata object could describe the source from which a CDN can acquire a piece of content. The GenericMetadata object is an atomic unit that can be referenced by HostMetadata or PathMetadata objects.

For example, if "example.com" is a content provider, a HostMatch object could include an entry for "example.com" with the URI of the associated HostMetadata object. The HostMetadata object for "example.com" describes the metadata properties which apply to "example.com" and could contain PathMatches for "example.com/movies/*" and "example.com/music/*", which in turn reference corresponding PathMetadata objects that contain the properties for those more specific URI paths. The PathMetadata object for "example.com/movies/*" describes the properties which apply to that URI path. It could also contain a PathMatch object for

"example.com/movies/hd/*" which would reference the corresponding PathMetadata object for the "example.com/movies/hd/" path prefix.

The relationships in Figure 1 are also represented in tabular format in Table 1 below.

Data Object	Objects it contains or references
HostIndex	0 or more HostMatch objects.
HostMatch	1 HostMetadata object.
HostMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.
PathMatch	1 PatternMatch object. 1 PathMetadata object.
PatternMatch	Does not contain or reference any other objects.
PathMetadata	0 or more PathMatch objects. 0 or more GenericMetadata objects.

Table 1: Relationships between CDNI Metadata Objects (Table Representation)

3.2. Generic CDNI Metadata Objects

The HostMetadata and PathMetadata objects contain other CDNI metadata objects that contain properties which describe how User Agent requests for content should be processed, for example where to acquire the content from, authorization rules that should be applied, geo-blocking restrictions, and so on. Each such CDNI metadata object is a specialization of a CDNI GenericMetadata object. The GenericMetadata object abstracts the basic information required for metadata override and metadata distribution, from the specifics of any given property (i.e., property semantics, enforcement options, etc.).

The GenericMetadata object defines the properties contained within it as well as whether or not the properties are "mandatory-to-enforce". If the dCDN does not understand or support a "mandatory-to-enforce" property, the dCDN MUST NOT serve the content. If the property is not "mandatory-to-enforce", then that GenericMetadata object can be safely ignored and the content request can be processed in accordance with the rest of the CDNI metadata.

Although a CDN MUST NOT serve content to a User Agent if a "mandatory-to-enforce" property cannot be enforced, it could still be "safe-to-redistribute" that metadata to another CDN without modification. For example, in the cascaded CDN case, a transit CDN (tCDN) could pass through "mandatory-to-enforce" metadata to a dCDN.

For metadata which does not require customization or translation (i.e., metadata that is "safe-to-redistribute"), the data representation received off the wire MAY be stored and redistributed without being understood or supported by the transit CDN. However, for metadata which requires translation, transparent redistribution of the uCDN metadata values might not be appropriate. Certain metadata can be safely, though perhaps not optimally, redistributed unmodified. For example, source acquisition address might not be optimal if transparently redistributed, but it might still work.

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

Transit CDNs MUST NOT change the value of "mandatory-to-enforce" or "safe-to-redistribute" when propagating metadata to a dCDN. Although a transit CDN can set the value of "incomprehensible" to true, a transit CDN MUST NOT change the value of "incomprehensible" from true to false.

Table 2 describes the action to be taken by a transit CDN (tCDN) for the different combinations of "mandatory-to-enforce" (MtE) and "safe-to-redistribute" (StR) properties, when the tCDN either does or does not understand the metadata in question:

MtE	StR	Metadata Understood by tCDN	Action
False	True	True	Can serve and redistribute.
False	True	False	Can serve and redistribute.
False	False	False	Can serve. MUST set "incomprehensible" to True when redistributing.
False	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	True	True	Can serve and redistribute.
True	True	False	MUST NOT serve but can redistribute.
True	False	True	Can serve. Can redistribute after transforming the metadata (if the CDN knows how to do so safely), otherwise MUST set "incomprehensible" to True when redistributing.
True	False	False	MUST NOT serve. MUST set "incomprehensible" to True when redistributing.

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

Table 3 describes the action to be taken by a dCDN for the different combinations of "mandatory-to-enforce" (MtE) and "incomprehensible" (Incomp) properties, when the dCDN either does or does not understand the metadata in question:

MtE	Incomp	Metadata Understood by dCDN	Action
False	False	True	Can serve.
False	True	True	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
False	False	False	Can serve.
False	True	False	Can serve but MUST NOT interpret/apply any metadata marked incomprehensible.
True	False	True	Can serve.
True	True	True	MUST NOT serve.
True	False	False	MUST NOT serve.
True	True	False	MUST NOT serve.

Table 3: Action to be taken by a dCDN for the different combinations of MtE and Incomp properties

3.3. Metadata Inheritance and Override

In the metadata object model, a HostMetadata object can contain multiple PathMetadata objects (via PathMatch objects). Each PathMetadata object can in turn contain other PathMetadata objects. HostMetadata and PathMetadata objects form an inheritance tree where each node in the tree inherits or overrides the property values set by its parent.

GenericMetadata objects of a given type override all GenericMetadata objects of the same type previously defined by any parent object in the tree. GenericMetadata objects of a given type previously defined by a parent object in the tree are inherited when no object of the same type is defined by the child object. For example, if HostMetadata for the host "example.com" contains GenericMetadata objects of type LocationACL and TimeWindowACL, while a PathMetadata object which applies to "example.com/movies/*" defines an alternate GenericMetadata object of type TimeWindowACL, then:

- the TimeWindowACL defined in the PathMetadata would override the TimeWindowACL defined in the HostMetadata for all User Agent requests for content under "example.com/movies/", and

- the LocationACL defined in the HostMetadata would be inherited for all User Agent requests for content under "example.com/movies/".

A single HostMetadata or PathMetadata object MUST NOT contain multiple GenericMetadata objects of the same type. If an array of GenericMetadata contains objects of duplicate types, the receiver MUST ignore all but the first object of each type.

4. CDNI Metadata objects

Section 4.1 provides the definitions of each metadata object type introduced in Section 3. These metadata objects are described as structural metadata objects as they provide the structure for host and URI path-based inheritance and identify which GenericMetadata objects apply to a given User Agent content request.

Section 4.2 provides the definitions for a base set of core metadata objects which can be contained within a GenericMetadata object. These metadata objects govern how User Agent requests for content are handled. GenericMetadata objects can contain other GenericMetadata as properties; these can be referred to as sub-objects). As with all CDNI metadata objects, the value of the GenericMetadata sub-objects can be either a complete serialized representation of the sub-object, or a Link object that contains a URI that can be dereferenced to retrieve the complete serialized representation of the property sub-object.

Section 6.5 discusses the ability to extend the base set of GenericMetadata objects specified in this document with additional standards-based or vendor specific GenericMetadata objects that might be defined in the future in separate documents.

dCDNs and tCDNs MUST support parsing of all CDNI metadata objects specified in this document. A dCDN does not have to implement the underlying functionality represented by non-structural GenericMetadata objects (though that might restrict the content that a given dCDN will be able to serve). uCDNs as generators of CDNI metadata only need to support generating the CDNI metadata that they need in order to express the policies required by the content they are describing. See Section 6.4 for more details on the specific encoding rules for CDNI metadata objects.

Note: In the following sections, the term "mandatory-to-specify" is used to convey which properties MUST be included for a given structural or GenericMetadata object. When mandatory-to-specify is specified as "Yes" for an individual property, it means that if the object containing that property is included in a metadata response, then the mandatory-to-specify property MUST also be included (directly or by reference) in the response, e.g., a HostMatch property object without a host to match against does not make sense,

therefore, the host property is mandatory-to-specify inside a HostMatch object.

4.1. Definitions of the CDNI structural metadata objects

Each of the sub-sections below describe the structural objects introduced in Section 3.1.

4.1.1. HostIndex

The HostIndex object is the entry point into the CDNI metadata hierarchy. It contains an array of HostMatch objects. An incoming content request is checked against the Hostname (or IP address) specified by each of the listed HostMatch objects to find the HostMatch object which applies to the request.

Property: hosts

Description: Array of HostMatch objects. Hosts (HostMatch objects) MUST be evaluated in the order they appear and the first HostMatch object that matches the content request being processed MUST be used.

Type: Array of HostMatch objects

Mandatory-to-Specify: Yes.

Example HostIndex object containing two HostMatch objects, where the first HostMatch object is embedded and the second HostMatch object is referenced:

```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "type": "MI.HostMatch",
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```

4.1.2. HostMatch

The HostMatch object contains a Hostname or IP address to match against content requests. The HostMatch object also contains a HostMetadata object to apply if a match is found.

Property: host

Description: Hostname or IP address and optional port to match against the requested host, i.e., the [RFC3986] host and port. In order for a Hostname or IP address in a content request to match the Hostname or IP address in the host property the value from the content request when converted to lowercase MUST be identical to the value of the host property when converted to lowercase. All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the IDNA encoding as per [RFC3490].

Type: Endpoint

Mandatory-to-Specify: Yes.

Property: host-metadata

Description: CDNI metadata to apply when delivering content that matches this host.

Type: HostMetadata

Mandatory-to-Specify: Yes.

Example HostMatch object with an embedded HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    <Properties of embedded HostMetadata object>
  }
}
```

Example HostMatch object referencing (via a Link object, see Section 4.3.1) a HostMetadata object:

```
{
  "host": "video.example.com",
  "host-metadata" : {
    "type": "MI.HostMetadata",
    "href": "https://metadata.ucdn.example/host1234"
  }
}
```

4.1.3. HostMetadata

A HostMetadata object contains the CDNI metadata properties for content served for a particular host (defined in the HostMatch object) and possibly child PathMatch objects.

Property: metadata

Description: Array of host related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example HostMetadata object containing a number of embedded GenericMetadata objects that will describe the default metadata for the host and an embedded PathMatch object that contains a path for which metadata exists that overrides the default metadata for the host:

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },

 ...

    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.4. PathMatch

A PathMatch object contains PatternMatch object with a path to match against a resource's URI path, as well as a PathMetadata object with GenericMetadata to apply if the resource's URI path matches the pattern within the PatternMatch object.

Property: path-pattern

Description: Pattern to match against the requested resource's URI path, i.e., against the [RFC3986] path-absolute.

Type: PatternMatch

Mandatory-to-Specify: Yes.

Property: path-metadata

Description: CDNI metadata to apply when delivering content that matches the associated PatternMatch.

Type: PathMetadata

Mandatory-to-Specify: Yes.

Example PathMatch object referencing the PathMetadata object to use for URIs that match the case-sensitive URI path pattern "/movies/*" (contained within an embedded PatternMatch object):

```
{
  "path-pattern": {
    "pattern": "/movies/*",
    "case-sensitive": true
  },
  "path-metadata": {
      "type": "MI.PathMetadata",
      "href": "https://metadata.ucdn.example/host1234/pathDCE"
  }
}
```

4.1.5. PatternMatch

A PatternMatch object contains the pattern string and flags that describe the pattern expression.

Property: pattern

Description: A pattern for string matching. The pattern can contain the wildcards * and ?, where * matches any sequence of characters (including the empty string) and ? matches exactly one character. The three literals $, * and ? should be escaped as $$, $* and $? (where $ is the designated escape character). All other characters are treated as literals.

Type: String

Mandatory-to-Specify: Yes.

Property: case-sensitive

Description: Flag indicating whether or not case-sensitive matching should be used. Note: Case-insensitivity applies to ALPHA characters in the URI path prior to percent-decoding [RFC3986].

Type: Boolean

Mandatory-to-Specify: No. Default is case-insensitive match.

Property: ignore-query-string

Description: Array of query parameters which should be ignored when searching for a pattern match. Matching against query

parameters to ignore MUST be case-insensitive. If all query parameters should be ignored then the list MUST be empty.

Type: Array of String

Mandatory-to-Specify: No. Default is to include query strings when matching.

Example PatternMatch object that matches the case-sensitive URI path pattern "/movies/*". All query parameters will be ignored when matching URIs requested from surrogates by content clients against this path pattern:

```
{
  "pattern": "/movies/*",
  "case-sensitive": true,
  "ignore-query-string": []
}
```

Example PatternMatch object that matches the case-sensitive URI path pattern "/movies/*". The query parameter "sessionid" will be ignored when matching URIs requested from surrogates by content clients against this path pattern:

```
{
  "pattern": "/movies/*",
  "case-sensitive": true,
  "ignore-query-string": ["sessionid"]
}
```

4.1.6. PathMetadata

A PathMetadata object contains the CDNI metadata properties for content requests that match against the associated URI path (defined in a PathMatch object).

Note that if DNS-based redirection is employed, then a dCDN will be unable to evaluate any metadata at the PathMetadata level or below because only the hostname of the content request is available at request routing time. dCDNs SHOULD still process all PathMetadata for the host before responding to the redirection request to detect if any unsupported metadata is specified. If any metadata not supported by the dCDN is marked as "mandatory-to-enforce", the dCDN SHOULD NOT accept the content redirection request, in order to avoid receiving content requests that it will not be able to satisfy/serve.

Property: metadata

Description: Array of path related metadata.

Type: Array of GenericMetadata objects

Mandatory-to-Specify: Yes.

Property: paths

Description: Path specific rules. Path patterns (PathMatch objects) MUST be evaluated in the order they appear and the first (and only the first) PathMatch object that matches the content request being processed MUST be used.

Type: Array of PathMatch objects

Mandatory-to-Specify: No.

Example PathMetadata object containing a number of embedded GenericMetadata objects that describe the metadata to apply for the URI path defined in the parent PathMatch object, as well as a more specific PathMatch object.

```
{
  "metadata": [
    {
      <Properties of 1st embedded GenericMetadata object>
    },
    {
      <Properties of 2nd embedded GenericMetadata object>
    },

 ...

    {
      <Properties of Nth embedded GenericMetadata object>
    }
  ],
  "paths": [
    {
      <Properties of embedded PathMatch object>
    }
  ]
}
```

4.1.7. GenericMetadata

A GenericMetadata object is a wrapper for managing individual CDNI metadata properties in an opaque manner.

Property: generic-metadata-type

Description: Case-insensitive CDNI metadata object type.

Type: String containing the CDNI Payload Type [RFC7736] of the object contained in the generic-metadata-value property (see Table 4).

Mandatory-to-Specify: Yes.

Property: generic-metadata-value

Description: CDNI metadata object.

Type: Format/Type is defined by the value of generic-metadata-type property above. Note: generic-metadata-values MUST NOT name any properties "href" (see Section 4.3.1).

Mandatory-to-Specify: Yes.

Property: mandatory-to-enforce

Description: Flag identifying whether or not the enforcement of the property metadata is required.

Type: Boolean

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

Property: safe-to-redistribute

Description: Flag identifying whether or not the property metadata can be safely redistributed without modification.

Type: Boolean

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

Property: incomprehensible

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

Type: Boolean

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

Example GenericMetadata object containing a metadata object that applies to the applicable URI path and/or host (within a parent PathMetadata and/or HostMetadata object, respectively):

```
{
  "mandatory-to-enforce": true,
  "safe-to-redistribute": true,
  "incomprehensible": false,
  "generic-metadata-type": <CDNI Payload Type of this metadata object>,
  "generic-metadata-value":
    {
      <Properties of this metadata object>
    }
}
```

4.2. Definitions of the initial set of CDNI Generic Metadata objects

The objects defined below are intended to be used in the GenericMetadata object generic-metadata-value field as defined in Section 4.1.7 and their generic-metadata-type property MUST be set to the appropriate CDNI Payload Type as defined in Table 4.

4.2.1. SourceMetadata

Source metadata provides the dCDN with information about content acquisition, i.e., how to contact an uCDN Surrogate or an Origin Server to obtain the content to be served. The sources are not necessarily the actual Origin Servers operated by the CSP but might be a set of Surrogates in the uCDN.

Property: sources

Description: Sources from which the dCDN can acquire content, listed in order of preference.

Type: Array of Source objects (see Section 4.2.1.1)

Mandatory-to-Specify: No. Default is to use static configuration, out-of-band from the metadata interface.

Example SourceMetadata object (which contains two Source objects) that describes which servers the dCDN should use for acquiring content for the applicable URI path and/or host:

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

4.2.1.1. Source

A Source object describes the source to be used by the dCDN for content acquisition (e.g., a Surrogate within the uCDN or an alternate Origin Server), the protocol to be used, and any authentication method to be used when contacting that source.

Endpoints within a Source object MUST be treated as equivalent/equal. A uCDN can specify an array of sources in preference order within a SourceMetadata object, and then for each preference ranked Source object, a uCDN can specify an array of endpoints that are equivalent (e.g., a pool of servers that are not behind a load balancer).

Property: acquisition-auth

Description: Authentication method to use when requesting content from this source.

Type: Auth (see Section 4.2.7)

Mandatory-to-Specify: No. Default is no authentication required.

Property: endpoints

Description: Origins from which the dCDN can acquire content. If multiple endpoints are specified they are all equal, i.e., the list is not in preference order (e.g., a pool of servers behind a load balancer).

Type: Array of Endpoint objects (See Section 4.3.3)

Mandatory-to-Specify: Yes.

Property: protocol

Description: Network retrieval protocol to use when requesting content from this source.

Type: Protocol (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Example Source object that describes a pair of endpoints (servers) the dCDN can use for acquiring content for the applicable host and/or URI path:

```
{
  "endpoints": [
    "a.service123.ucdn.example",
    "b.service123.ucdn.example"
  ],
  "protocol": "http/1.1"
}
```

4.2.2. LocationACL Metadata

LocationACL metadata defines which locations a User Agent needs to be in, in order to be able to receive the associated content.

A LocationACL which does not include a locations property results in an action of allow all, meaning that delivery can be performed regardless of the User Agent’s location, otherwise a CDN MUST take the action from the first footprint to match against the User Agent’s location. If two or more footprints overlap, the first footprint that matches against the User Agent’s location determines the action a CDN MUST take. If the locations property is included but is empty,

or if none of the listed footprints matches the User Agent's location, then the result is an action of deny.

Although the LocationACL, TimeWindowACL (see Section 4.2.3), and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: locations

Description: Access control list which allows or denies (blocks) delivery based on the User Agent's location.

Type: Array of LocationRule objects (see Section 4.2.2.1)

Mandatory-to-Specify: No. Default is allow all locations.

Example LocationACL object that allows the dCDN to deliver content to any location/IP address:

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

Example LocationACL object (which contains a LocationRule object which itself contains a Footprint object) that only allows the dCDN to deliver content to User Agents in the USA:

```
{
  "generic-metadata-type": "MI.LocationACL",
  "generic-metadata-value":
    {
      "locations": [
        {
          "action": "allow",
          "footprints": [
            {
              "footprint-type": "countrycode",
              "footprint-value": ["us"]
            }
          ]
        }
      ]
    }
}
```

4.2.2.1. LocationRule

A LocationRule contains or references an array of Footprint objects and the corresponding action.

Property: footprints

Description: Array of footprints to which the rule applies.

Type: Array of Footprint objects (see Section 4.2.2.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies locations to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example LocationRule object (which contains a Footprint object) that allows the dCDN to deliver content to clients in the USA:

```
{
  "action": "allow",
  "footprints": [
    {
      "footprint-type": "countrycode",
      "footprint-value": ["us"]
    }
  ]
}
```

4.2.2.2. Footprint

A Footprint object describes the footprint to which a LocationRule can be applied to, e.g., an IPv4 address range or a geographic location.

Property: footprint-type

Description: Registered footprint type (see Section 7.2). The footprint types specified by this document are: "ipv4cidr" (IPv4CIDR, see Section 4.3.5), "ipv6cidr" (IPv6CIDR, see Section 4.3.6), "asn" (Autonomous System Number, see Section 4.3.7) and "countrycode" (Country Code, see Section 4.3.8).

Type: Lowercase String

Mandatory-to-Specify: Yes.

Property: footprint-value

Description: Array of footprint values conforming to the specification associated with the registered footprint type. Footprint values can be simple strings (e.g., IPv4CIDR, IPv6CIDR, ASN, and CountryCode), however, other Footprint objects can be defined in the future, along with a more complex encoding (e.g., GPS coordinate tuples).

Type: Array of footprints

Mandatory-to-Specify: Yes.

Example Footprint object describing a footprint covering the USA:

```
{
  "footprint-type": "countrycode",
  "footprint-value": ["us"]
}
```

Example Footprint object describing a footprint covering the IP address ranges 192.0.2.0/24 and 198.51.100.0/24:

```
{
  "footprint-type": "ipv4cidr",
  "footprint-value": ["192.0.2.0/24", "198.51.100.0/24"]
}
```

Example Footprint object describing a footprint covering the IP address ranges 2001:db8::/32:

```
{
  "footprint-type": "ipv6cidr",
  "footprint-value": ["2001:db8::/32"]
}
```

Example Footprint object describing a footprint covering the autonomous system 64496:

```
{
  "footprint-type": "asn",
  "footprint-value": ["as64496"]
}
```

4.2.3. TimeWindowACL

TimeWindowACL metadata defines time-based restrictions.

A TimeWindowACL which does not include a times property results in an action of allow all, meaning that delivery can be performed regardless of the time of the User Agent's request, otherwise a CDN MUST take the action from the first window to match against the current time. If two or more windows overlap, the first window that matches against the current time determines the action a CDN MUST take. If the times property is included but is empty, or if none of the listed windows matches the current time, then the result is an action of deny.

Although the LocationACL (see Section 4.2.2), TimeWindowACL, and ProtocolACL (see Section 4.2.4) are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the LocationACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: times

Description: Access control list which allows or denies (blocks) delivery based on the time of a User Agent's request.

Type: Array of TimeWindowRule objects (see Section 4.2.3.1)

Mandatory-to-Specify: No. Default is allow all time windows.

Example TimeWIndowACL object (which contains a TimeWindowRule object which itself contains a TimeWIndow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

```
{
  "generic-metadata-type": "MI.TimeWindowACL",
  "generic-metadata-value":
    {
      "times": [
        {
          "action": "allow",
          "windows": [
            {
              "start": 946717200,
              "end": 946746000
            }
          ]
        }
      ]
    }
}
```

4.2.3.1. TimeWindowRule

A TimeWindowRule contains or references an array of TimeWindow objects and the corresponding action.

Property: windows

Description: Array of time windows to which the rule applies.

Type: Array of TimeWindow objects (see Section 4.2.3.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies time windows to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example TimeWIndowRule object (which contains a TimeWIndow object) that only allows the dCDN to deliver content to clients between 09:00 01/01/2000 UTC and 17:00 01/01/2000 UTC:

```
{
  "action": "allow",
  "windows": [
    {
      "start": 946717200,
      "end": 946746000
    }
  ]
}
```

4.2.3.2. TimeWindow

A TimeWindow object describes a time range which can be applied by an TimeWindowACL, e.g., start 946717200 (i.e., 09:00 01/01/2000 UTC), end: 946746000 (i.e., 17:00 01/01/2000 UTC).

Property: start

Description: The start time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Property: end

Description: The end time of the window.

Type: Time (see Section 4.3.4)

Mandatory-to-Specify: Yes.

Example TimeWIndow object that describes a time window from 09:00 01/01/2000 UTC to 17:00 01/01/2000 UTC:

```
{
  "start": 946717200,
  "end": 946746000
}
```

4.2.4. ProtocolACL Metadata

ProtocolACL metadata defines delivery protocol restrictions.

A ProtocolACL which does not include a protocol-acl property results in an action of allow all, meaning that delivery can be performed regardless of the protocol in the User Agent's request, otherwise a CDN MUST take the action from the first protocol to match against the request protocol. If two or more request protocols overlap, the first protocol that matches the request protocol determines the action a CDN MUST take. If the protocol-acl property is included but is empty, or if none of the listed protocol matches the request protocol, then the result is an action of deny.

Although the LocationACL, TimeWindowACL, and ProtocolACL are independent GenericMetadata objects, they can provide conflicting information to a dCDN, e.g., a content request which is simultaneously allowed based on the ProtocolACL and denied based on the TimeWindowACL. The dCDN MUST use the logical AND of all ACLs (where 'allow' is true and 'deny' is false) to determine whether or not a request should be allowed.

Property: protocol-acl

Description: Description: Access control list which allows or denies (blocks) delivery based on delivery protocol.

Type: Array of ProtocolRule objects (see Section 4.2.4.1)

Mandatory-to-Specify: No. Default is allow all protocols.

Example ProtocolACL object (which contains a ProtocolRule object) that only allows the dCDN to deliver content using HTTP/1.1:

```
{
  "generic-metadata-type": "MI.ProtocolACL",
  "generic-metadata-value":
    {
      "protocol-acl": [
        {
          "action": "allow",
          "protocols": ["http/1.1"]
        }
      ]
    }
}
```

4.2.4.1. ProtocolRule

A ProtocolRule contains or references an array of Protocol objects and the corresponding action.

Property: protocols

Description: Array of protocols to which the rule applies.

Type: Array of Protocols (see Section 4.3.2)

Mandatory-to-Specify: Yes.

Property: action

Description: Defines whether the rule specifies protocols to allow or deny.

Type: Enumeration [allow|deny] encoded as a lowercase string

Mandatory-to-Specify: No. Default is deny.

Example ProtocolRule object (which contains a ProtocolRule object) that allows the dCDN to deliver content using HTTP/1.1:

```
{
  "action": "allow",
  "protocols": ["http/1.1"]
}
```

4.2.5. DeliveryAuthorization Metadata

Delivery Authorization defines authorization methods for the delivery of content to User Agents.

Property: delivery-auth-methods

Description: Options for authorizing content requests. Delivery for a content request is authorized if any of the authorization methods in the list is satisfied for that request.

Type: Array of Auth objects (see Section 4.2.7)

Mandatory-to-Specify: No. Default is no authorization required.

Example DeliveryAuthorization object (which contains an Auth object):

```
{
  "generic-metadata-type": "MI.DeliveryAuthorization",
  "generic-metadata-value":
    {
      "delivery-auth-methods": [
        {
          "auth-type": <CDNI Payload Type of this Auth object>,
          "auth-value":
          {
            <Properties of this Auth object>
          }
        }
      ]
    }
}
```

4.2.6. Cache

A Cache object describes the cache control parameters to be applied to the content by intermediate caches.

Property: ignore-query-string

Description: Allows a Surrogate to ignore URI query string parameters [RFC3986] when comparing the requested URI against the URIs in its cache for equivalence. Matching query parameters to ignore MUST be case-insensitive. Each query parameter to ignore is specified in the list. If all query parameters should be ignored, then the list MUST be specified and MUST be empty.

Type: Array of String

Mandatory-to-Specify: No. Default is to consider query string parameters when comparing URIs.

Example Cache object that instructs the dCDN to ignore all query parameters:

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "ignore-query-string": []
  }
}
```

Example Cache object that instructs the dCDN to ignore the (case-insensitive) query parameters named "sessionid" and "random":

```
{
  "generic-metadata-type": "MI.Cache",
  "generic-metadata-value":
  {
    "ignore-query-string": ["sessionid", "random"]
  }
}
```

4.2.7. Auth

An Auth object defines authentication and authorization methods to be used during content acquisition and content delivery, respectively.

Note: This document does not define any Auth methods. Individual Auth methods are being defined separately (e.g., URI Signing [I-D.ietf-cdni-uri-signing]). The GenericMetadata which contain Auth objects is defined herein for convenience and so as not to be specific to any particular Auth method.

Property: auth-type

Description: Auth type (The CDNI Payload Type [RFC7736] of the GenericMetadata object contained in the auth-value property).

Type: String

Mandatory-to-Specify: Yes.

Property: auth-value

Description: An object conforming to the specification associated with the Auth type.

Type: GenericMetadata Object

Mandatory-to-Specify: Yes.

Example Auth object:

```
{
  "generic-metadata-type": "MI.Auth",
  "generic-metadata-value":
  {
    "auth-type": <CDNI Payload Type of this Auth object>,
    "auth-value":
      {
        <Properties of this Auth object>
      }
  }
}
```

4.2.8. Grouping

A Grouping object identifies a group of content to which a given asset belongs.

Property: ccid

Description: Content Collection identifier for an application-specific purpose such as logging aggregation.

Type: String

Mandatory-to-Specify: No. Default is an empty string.

Example Grouping object that specifies a Content Collection Identifier for the content associated with the Grouping object's parent HostMetadata and PathMetadata:

```
{
  "generic-metadata-type": "MI.Grouping",
  "generic-metadata-value":
  {
    "ccid": "ABCD"
  }
}
```

4.3. CDNI Metadata Simple Data Type Descriptions

This section describes the simple data types that are used for properties of CDNI metadata objects.

4.3.1. Link

A Link object can be used in place of any of the objects or properties described above. Link objects can be used to avoid duplication if the same metadata information is repeated within the

metadata tree. When a Link object replaces another object, its href property is set to the URI of the resource and its type property is set to the CDNI Payload Type of the object it is replacing.

dCDNs can detect the presence of a Link object by detecting the presence of a property named "href" within the object. This means that GenericMetadata types MUST NOT contain a property named "href" because doing so would conflict with the ability for dCDNs to detect Link objects being used to reference a GenericMetadata object.

Property: href

Description: The URI of the addressable object being referenced.

Type: String

Mandatory-to-Specify: Yes.

Property: type

Description: The type of the object being referenced.

Type: String

Mandatory-to-Specify: No. If the container specifies the type (e.g., the HostIndex object contains an array of HostMatch objects, so a Link object in the list of HostMatch objects must reference a HostMatch), then it is not necessary to explicitly specify a type.

Example Link object referencing a HostMatch object:

```
{
  "type": "MI.HostMatch",
  "href": "https://metadata.ucdn.example/hostmatch1234"
}
```

Example Link object referencing a HostMatch object, without an explicit type, inside a HostIndex object:

```
{
  "hosts": [
    {
      <Properties of embedded HostMatch object>
    },
    {
      "href": "https://metadata.ucdn.example/hostmatch1234"
    }
  ]
}
```

4.3.2. Protocol

Protocol objects are used to specify registered protocols for content acquisition or delivery (see Section 7.3).

Type: String

Example:

"http/1.1"

4.3.3. Endpoint

A Hostname (with optional port) or an IP address (with optional port).

All implementations MUST support IPv4 addresses encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. IPv6 addresses MUST be encoded in one of the IPv6 address formats specified in [RFC5952] although receivers MUST support all IPv6 address formats specified in [RFC4291]. Hostnames MUST conform to the Domain Name System (DNS) syntax defined in [RFC1034] and [RFC1123]. Internationalized Domain Names (IDN) must first be transformed to the IDNA encoding as per [RFC3490].

Type: String

Example Hostname:

"metadata.ucdn.example"

Example IPv4 address:

"192.0.2.1"

Example IPv6 address (with port number):

"[2001:db8::1]:81"

4.3.4. Time

A time value expressed in seconds since the Unix epoch (i.e., zero hours, zero minutes, zero seconds, on January 1, 1970) Coordinated Universal Time (UTC) [POSIX].

Type: Integer

Example Time representing 09:00:00 01/01/2000 UTC:

946717200

4.3.5. IPv4CIDR

An IPv4address CIDR block encoded as specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv4 CIDR notation). Single IP addresses can be expressed as /32.

Type: String

Example IPv4 CIDR:

"192.0.2.0/24"

4.3.6. IPv6CIDR

An IPv6address CIDR block encoded in one of the IPv6 address formats specified in [RFC5952] followed by a / followed by an unsigned integer representing the leading bits of the routing prefix (i.e., IPv6 CIDR notation). Single IP addresses can be expressed as /128.

Type: String

Example IPv6 CIDR:

"2001:db8::/32"

4.3.7. ASN

An Autonomous System Number encoded as a string consisting of the characters "as" (in lowercase) followed by the Autonomous System number [RFC6793].

Type: String

Example ASN:

"as64496"

4.3.8. CountryCode

An ISO 3166-1 alpha-2 code [ISO3166-1] in lowercase.

Type: String

Example Country Code representing the USA:

"us"

5. CDNI Metadata Capabilities

CDNI metadata is used to convey information pertaining to content delivery from uCDN to dCDN. For optional metadata, it can be useful for the uCDN to know if the dCDN supports the underlying functionality described by the metadata, prior to delegating any content requests to the dCDN. If some metadata is "mandatory-to-enforce", and the dCDN does not support it, any delegated requests for content that requires that metadata will fail. The uCDN will likely want to avoid delegating those requests to that dCDN. Likewise, for any metadata which might be assigned optional values, it could be useful for the uCDN to know which values a dCDN supports, prior to delegating any content requests to that dCDN. If the optional value assigned to a given piece of content's metadata is not supported by the dCDN, any delegated requests for that content can fail, so again the uCDN is likely to want to avoid delegating those requests to that dCDN.

The CDNI Footprint and Capabilities Interface (FCI) provides a means of advertising capabilities from dCDN to uCDN [RFC7336]. Support for optional metadata types and values can be advertised using the FCI.

6. CDNI Metadata interface

This section specifies an interface to enable a dCDN to retrieve CDNI metadata objects from a uCDN.

The interface can be used by a dCDN to retrieve CDNI metadata objects either:

- Dynamically as required by the dCDN to process received requests. For example in response to a query from an uCDN over the CDNI Request Routing Redirection interface (RI)

[I-D.ietf-cdni-redirection] or in response to receiving a request for content from a User Agent. Or;

- In advance of being required. For example in the case of pre-positioned CDNI metadata acquisition, initiated through the "CDNI Control interface / Triggers" (CI/T) interface [I-D.ietf-cdni-control-triggers].

The CDNI metadata interface is built on the principles of HTTP web services. In particular, this means that requests and responses over the interface are built around the transfer of representations of hyperlinked resources. A resource in the context of the CDNI metadata interface is any object in the object model (as described in Section 3 and Section 4).

To retrieve CDNI metadata, a CDNI metadata client (i.e., a client in the dCDN) first makes a HTTP GET request for the URI of the HostIndex which provides the CDNI metadata client with an array of Hostnames for which the uCDN can delegate content delivery to the dCDN. The CDNI metadata client can then obtain any other CDNI metadata objects by making a HTTP GET requests for any linked metadata objects it requires.

CDNI metadata servers (i.e., servers in the uCDN) are free to assign whatever structure they desire to the URIs for CDNI metadata objects and CDNI metadata clients MUST NOT make any assumptions regarding the structure of CDNI metadata URIs or the mapping between CDNI metadata objects and their associated URIs. Therefore any URIs present in the examples in this document are purely illustrative and are not intended to impose a definitive structure on CDNI metadata interface implementations.

6.1. Transport

The CDNI metadata interface uses HTTP as the underlying protocol transport [RFC7230].

The HTTP Method in the request defines the operation the request would like to perform. A server implementation of the CDNI metadata interface MUST support the HTTP GET and HEAD methods.

The corresponding HTTP Response returns the status of the operation in the HTTP Status Code and returns the current representation of the resource (if appropriate) in the Response Body. HTTP Responses that contain a response body SHOULD include an ETag to enable validation of cached versions of returned resources.

The CDNI metadata interface specified in this document is a read-only interface. A server implementation of the CDNI metadata interface MUST reject attempts by a CDNI metadata client to update the CDNI metadata, including all requests with HTTP Method PUT, POST or DELETE.

As the CDNI metadata interface builds on top of HTTP, CDNI metadata server implementations MAY make use of any HTTP feature when implementing the CDNI metadata interface, for example, a CDNI metadata server MAY make use of HTTP's caching mechanisms to indicate that the returned response/representation can be reused without re-contacting the CDNI metadata server.

6.2. Retrieval of CDNI Metadata resources

In the general case, a CDNI metadata server makes CDNI metadata objects available via a unique URIs and thus, in order to retrieve CDNI metadata, a CDNI metadata client first makes a HTTP GET request for the URI of the HostIndex which provides an array of Hostnames for which the uCDN can delegate content delivery to the dCDN.

In order to retrieve the CDNI metadata for a particular request the CDNI metadata client processes the received HostIndex object and finds the corresponding HostMetadata entry (by matching the hostname in the request against the hostnames listed in the HostMatch objects). If the HostMetadata is linked (rather than embedded), the CDNI metadata client then makes a GET request for the URI specified in the href property of the Link object which points to the HostMetadata object itself.

In order to retrieve the most specific metadata for a particular request, the CDNI metadata client inspects the HostMetadata for references to more specific PathMetadata objects (by matching the URI path in the request against the path-patterns in any PathMatch objects listed in the HostMetadata object). If any PathMetadata are found to match (and are linked rather than embedded), the CDNI metadata client makes another GET request for the PathMetadata. Each PathMetadata object can also include references to yet more specific metadata. If this is the case, the CDNI metadata client continues requesting PathMatch and PathMetadata objects recursively. The CDNI metadata client repeats this approach of processing metadata objects and retrieving (via HTTP GETs) any linked objects until it has all the metadata objects it requires in order to process the redirection request from an uCDN or the content request from a User Agent.

In cases where a dCDN is not able to retrieve the entire set of CDNI metadata associated with a User Agent request, or it has retrieved that metadata but it is stale according to standard HTTP caching

rules and cannot be revalidated, for example because the uCDN is unreachable or returns a HTTP 4xx or 5xx status in response to some or all of the dCDN's CDNI metadata requests, the dCDN MUST NOT serve the requested content.

Where a dCDN is interconnected with multiple uCDNs, the dCDN needs to determine which uCDN's CDNI metadata should be used to handle a particular User Agent request.

When application level redirection (e.g., HTTP 302 redirects) is being used between CDNs, it is expected that the dCDN will be able to determine the uCDN that redirected a particular request from information contained in the received request (e.g., via the URI). With knowledge of which uCDN routed the request, the dCDN can choose the correct uCDN from which to obtain the HostIndex. Note that the HostIndexes served by each uCDN can be unique.

In the case of DNS redirection there is not always sufficient information carried in the DNS request from User Agents to determine the uCDN that redirected a particular request (e.g., when content from a given host is redirected to a given dCDN by more than one uCDN) and therefore dCDNs will have to apply local policy when deciding which uCDN's metadata to apply.

6.3. Bootstrapping

The URI for the HostIndex object of a given uCDN needs to be configured in the dCDN. All other objects/resources are then discoverable from the HostIndex object by following any links in the HostIndex object and through the referenced HostMetadata and PathMetadata objects and their GenericMetadata sub-objects.

Manual configuration of the URI for the HostIndex object is outside the scope of this document.

6.4. Encoding

CDNI metadata objects MUST be encoded as I-JSON objects [RFC7493] containing a dictionary of (key,value) pairs where the keys are the property names and the values are the associated property values.

The keys of the dictionary are the names of the properties associated with the object and are therefore dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource). Likewise, the values associated with each property (dictionary key) are dependent on the specific object being encoded (i.e., dependent on the CDNI Payload Type of the returned resource).

Dictionary keys (properties) in I-JSON are case sensitive. By convention, any dictionary key (property) defined by this document (for example, the names of CDNI metadata object properties) MUST be lowercase.

6.5. Extensibility

The set of GenericMetadata objects can be extended with additional (standards based or vendor specific) metadata objects through the specification of new GenericMetadata objects. The GenericMetadata object defined in Section 4.1.7 specifies a type field and a type-specific value field that allows any metadata to be included in either the HostMetadata or PathMetadata arrays.

As with the initial GenericMetadata types defined in Section 4.2, future GenericMetadata types MUST specify the information necessary for constructing and decoding the GenericMetadata object.

Any document which defines a new GenericMetadata type MUST:

1. Specify and register the CDNI Payload Type [RFC7736] used to identify the new GenericMetadata type being specified.

2. Define the set of properties associated with the new GenericMetadata object. GenericMetadata MUST NOT contain a property named "href" because doing so would conflict with the ability to detect Link objects (see Section 4.3.1).

3. Define a name, description, type, and whether or not the property is mandatory-to-specify.

4. Describe the semantics of the new type including its purpose and example of a use case to which it applies including an example encoded in I-JSON.

5. Describe the security and privacy consequences, for both the user-agent and the CDN, of the new GenericMetadata object.

6. Describe any relation to, conflict with, or obsolescence of other existing CDNI metadata objects.

Note: In the case of vendor specific extensions, vendor-identifying CDNI Payload Type names will decrease the possibility of GenericMetadata type collisions.

6.6. Metadata Enforcement

At any given time, the set of GenericMetadata types supported by the uCDN might not match the set of GenericMetadata types supported by the dCDN.

In cases where a uCDN sends metadata containing a GenericMetadata type that a dCDN does not support, the dCDN MUST enforce the semantics of the "mandatory-to-enforce" property. If a dCDN does not understand or is unable to perform the functions associated with any "mandatory-to-enforce" metadata, the dCDN MUST NOT service any requests for the corresponding content.

Note: Ideally, uCDNs would not delegate content requests to a dCDN that does not support the "mandatory-to-enforce" metadata associated with the content being requested. However, even if the uCDN has a priori knowledge of the metadata supported by the dCDN (e.g., via the FCI or through out-of-band negotiation between CDN operators), metadata support can fluctuate or be inconsistent (e.g., due to mis-communication, mis-configuration, or temporary outage). Thus, the dCDN MUST always evaluate all metadata associated with redirection and content requests and reject any requests where "mandatory-to-enforce" metadata associated with the content cannot be enforced.

6.7. Metadata Conflicts

It is possible that new metadata definitions will obsolete or conflict with existing GenericMetadata (e.g., a future revision of the CDNI metadata interface could redefine the Auth GenericMetadata object or a custom vendor extension could implement an alternate Auth metadata option). If multiple metadata (e.g., MI.Auth.v2, vendor1.Auth, and vendor2.Auth) all conflict with an existing GenericMetadata object (i.e., MI.Auth) and all are marked as "mandatory-to-enforce", it could be ambiguous which metadata should be applied, especially if the functionality of the metadata overlap.

As described in Section 3.3, metadata override only applies to metadata objects of the same exact type found in HostMetadata and nested PathMetadata structures. The CDNI metadata interface does not support enforcement of dependencies between different metadata types. It is the responsibility of the CSP and the CDN operators to ensure that metadata assigned to a given piece of content do not conflict.

Note: Because metadata is inherently ordered in HostMetadata and PathMetadata arrays, as well as in the PathMatch hierarchy, multiple conflicting metadata types MAY be used, however, metadata hierarchies SHOULD ensure that independent PathMatch root objects are used to prevent ambiguous or conflicting metadata definitions.

6.8. Versioning

The version of CDNI metadata objects is conveyed inside the CDNI Payload Type that is included in either the HTTP Content-Type header, for example: "Content-Type: application/cdni; ptype=MI.HostIndex", when retrieved via a link, or in the link type (Section 4.3.1), generic-metadata-type (Section 4.1.7), or auth-type (Section 4.2.7) properties in the JSON payload. The CDNI Payload Type uniquely identifies the specification defining that object, including any relation to, conflicts with, or obsolescence of other metadata. There is no explicit version mapping requirement, however, for ease of understanding, metadata creators SHOULD make new versions of metadata easily visible via the CDNI Payload Type, e.g., by appending a version string. Note: A version string is optional on the first version, e.g., MI.HostIndex, but could be added for subsequent versions, e.g., MI.HostIndex.v2, MI.HostIndex.v3, etc.

In some cases, structural metadata Section 4.1 may be included without explicit type designation. [Ed. Note: There are a couple options here: 1) require that all structural metadata is referenced via link, so that the link type is present, or 2) require that all structural metadata be versioned together, so that one can always tell the version of an embedded structural object based on the version of the outer-most structural object, referenced in the Content-Type.]

HTTP requests sent to a metadata server SHOULD include an Accept header with the CDNI Payload Type of the expected object. Metadata clients can specify multiple CDNI Payload Types in the Accept header, for example, if a metadata client is capable of processing two different versions of the same type of object (defined by different CDNI Payload Types) it might decide to include both in the Accept header.

6.9. Media Types

All CDNI metadata objects use the Media Type "application/cdni". The CDNI Payload Type for each object then contains the object name of that object as defined by this document, prefixed with "MI.". Table 4 lists the CDNI Payload Type for the metadata objects (resources) specified in this document.

Data Object	CDNI Payload Type
HostIndex	MI.HostIndex
HostMatch	MI.HostMatch
HostMetadata	MI.HostMetadata
PathMatch	MI.PathMatch
PatternMatch	MI.PatternMatch
PathMetadata	MI.PathMetadata
SourceMetadata	MI.SourceMetadata
Source	MI.Source
LocationACL	MI.LocationACL
LocationRule	MI.LocationRule
Footprint	MI.Footprint
TimeWindowACL	MI.TimeWindowACL
TimeWindowRule	MI.TimeWindowRule
TimeWindow	MI.TimeWindow
ProtocolACL	MI.ProtocolACL
ProtocolRule	MI.ProtocolRule
DeliveryAuthorization	MI.DeliveryAuthorization
Cache	MI.Cache
Auth	MI.Auth
Grouping	MI.Grouping

Table 4: CDNI Payload Types for CDNI Metadata objects

6.10. Complete CDNI Metadata Example

A dCDN requests the HostIndex and receive the following object with a CDNI payload type of "MI.HostIndex":

```
{
  "hosts": [
    {
      "host": "video.example.com",
      "host-metadata" : {
        "type": "MI.HostMetadata",
        "href": "https://metadata.ucdn.example/host1234"
      }
    },
    {
      "host": "images.example.com",
      "host-metadata" : {
        "type": "MI.HostMetadata",
        "href": "https://metadata.ucdn.example/host5678"
      }
    }
  ]
}
```

If the incoming request has a Host header with "video.example.com" then the dCDN would fetch the HostMetadata object from "https://metadata.ucdn.example/host1234" expecting a CDNI payload type of "MI.HostMetadata":

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.SourceMetadata",
      "generic-metadata-value": {
        "sources": [
          {
            "endpoint": ["acq1.ucdn.example"],
            "protocol": "http/1.1"
          },
          {
            "endpoint": ["acq2.ucdn.example"],
            "protocol": "http/1.1"
          }
        ]
      }
    },
    {
      "generic-metadata-type": "MI.LocationACL",
      "generic-metadata-value": {
        "locations": [
          {
            "footprints": [
              {
```

```
                  "footprint-type": "ipv4cidr",
                  "footprint-value": ["192.0.2.0/24"]
                },
                {
                  "footprint-type": "ipv6cidr",
                  "footprint-value": ["2001:db8::/32"]
                },
                {
                  "footprint-type": "countrycode",
                  "footprint-value": ["us"]
                },
                {
                  "footprint-type": "asn",
                  "footprint-value": ["as64496"]
                }
              ],
              "action": "deny"
            }
          ]
        }
      },
      {
        "generic-metadata-type": "MI.ProtocolACL",
        "generic-metadata-value": {
          "protocol-acl": [
            {
              "protocols": [
                "http/1.1"
              ],
              "action": "allow"
            }
          ]
        }
      }
    ],
    "paths": [
      {
        "path-pattern": {
          "pattern": "/video/trailers/*"
        },
        "path-metadata": {
          "type": "MI.PathMetadata",
          "href": "https://metadata.ucdn.example/host1234/pathABC"
        }
      },
      {
        "path-pattern": {
          "pattern": "/video/movies/*"
```

```
        },
        "path-metadata": {
          "type": "MI.PathMetadata",
          "href": "https://metadata.ucdn.example/host1234/pathDEF"
        }
      }
    ]
  }
```

Suppose the path of the requested resource matches the "/video/movies/*" pattern, the next metadata requested would be for "https://metadata.ucdn.example/host1234/pathDCE" with an expected CDNI payload type of "MI.PathMetadata":

```
{
  "metadata": [],
  "paths": [
    {
      "path-pattern": {
        "pattern": "/videos/movies/hd/*"
      },
      "path-metadata": {
        "type": "MI.PathMetadata",
        "href":
          "https://metadata.ucdn.example/host1234/pathDEF/path123"
      }
    }
  ]
}
```

Finally, if the path of the requested resource also matches the "/videos/movies/hd/*" pattern, the dCDN would also fetch the following object from "https://metadata.ucdn.example/host1234/pathDEF/path123" with CDNI payload type "MI.PathMetadata":

```
{
  "metadata": [
    {
      "generic-metadata-type": "MI.TimeWindowACL",
      "generic-metadata-value": {
        "times": [
          "windows": [
            {
              "start": "1213948800",
              "end": "1327393200"
            }
          ],
          "action": "allow"
        ]
      }
    }
  ]
}
```

The final set of metadata which applies to the requested resource includes a SourceMetadata, a LocationACL, a ProtocolACL, and a TimeWindowACL.

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:

Payload Type	Specification
MI.HostIndex	RFCthis
MI.HostMatch	RFCthis
MI.HostMetadata	RFCthis
MI.PathMatch	RFCthis
MI.PatternMatch	RFCthis
MI.PathMetadata	RFCthis
MI.SourceMetadata	RFCthis
MI.Source	RFCthis
MI.LocationACL	RFCthis
MI.LocationRule	RFCthis
MI.Footprint	RFCthis
MI.TimeWindowACL	RFCthis
MI.TimeWindowRule	RFCthis
MI.TimeWindow	RFCthis
MI.ProtocolACL	RFCthis
MI.ProtocolRule	RFCthis
MI.DeliveryAuthorization	RFCthis
MI.Cache	RFCthis
MI.Auth	RFCthis
MI.Grouping	RFCthis

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

7.1.1. CDNI MI HostIndex Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.1

7.1.2. CDNI MI HostMatch Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.2

7.1.3. CDNI MI HostMetadata Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.3

7.1.4. CDNI MI PathMatch Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.4

7.1.5. CDNI MI PatternMatch Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.5

7.1.6. CDNI MI PathMetadata Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.1.6

7.1.7. CDNI MI SourceMetadata Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.1

7.1.8. CDNI MI Source Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.1.1

7.1.9. CDNI MI LocationACL Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.2

7.1.10. CDNI MI LocationRule Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.2.1

7.1.11. CDNI MI Footprint Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.2.2

7.1.12. CDNI MI TimeWindowACL Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.3

7.1.13. CDNI MI TimeWindowRule Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.3.1

7.1.14. CDNI MI TimeWindow Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.3.2

7.1.15. CDNI MI ProtocolACL Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.4

7.1.16. CDNI MI ProtocolRule Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.4.1

7.1.17. CDNI MI DeliveryAuthorization Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.5

7.1.18. CDNI MI Cache Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.6

7.1.19. CDNI MI Auth Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.7

7.1.20. CDNI MI Grouping Payload Type

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

Interface: MI/FCI

Encoding: see Section 4.2.8

7.2. CDNI Metadata Footprint Types Registry

The IANA is requested to create a new "CDNI Metadata Footprint Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Footprint Types" namespace defines the valid Footprint object type values used by the Footprint object in Section 4.2.2.2. Additions to the Footprint type namespace conform to the "Specification Required" policy as defined in [RFC5226]. The designated expert will verify that new type definitions do not duplicate existing type definitions and prevent gratuitous additions to the namespace. New registrations are required to provide a clear description of how to interpret new footprint types.

The following table defines the initial Footprint Registry values:

Footprint Type	Description	Specification
ipv4cidr	IPv4 CIDR address block	RFCthis
ipv6cidr	IPv6 CIDR address block	RFCthis
asn	Autonomous System (AS) Number	RFCthis
countrycode	ISO 3166-1 alpha-2 code	RFCthis

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

7.3. CDNI Metadata Protocol Types Registry

The IANA is requested to create a new "CDNI Metadata Protocol Types" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI Metadata Protocol Types" namespace defines the valid Protocol object values in Section 4.3.2, used by the SourceMetadata and ProtocolACL objects. Additions to the Protocol namespace conform to the "Specification Required" policy as defined in [RFC5226], where the specification defines the Protocol Type and the protocol to which it is associated. The designated expert will verify that new protocol definitions do not duplicate existing protocol definitions and prevent gratuitous additions to the namespace.

The following table defines the initial Protocol values corresponding to the HTTP and HTTPS protocols:

Protocol Type	Description	Type Specification	Protocol Specifications
http/1.1	Hypertext Transfer Protocol -- HTTP/1.1	RFCthis	RFC7230
https/1.1	HTTP/1.1 Over TLS	RFCthis	RFC7230, RFC2818

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

8. Security Considerations

8.1. Authentication and Integrity

A malicious metadata server, proxy server, or attacker, impersonating an authentic uCDN metadata interface without being detected, could provide false metadata to a dCDN that either:

- Denies service for one or more pieces of content to one or more User Agents;
- Directs dCDNs to contact malicious origin servers instead of the actual origin servers, and substitute legitimate content with malware or slanderous alternate content; or
- Removes delivery restrictions (e.g., LocationACL, TimeWindowACL, ProtocolACL, or Auth metadata), allowing access to content that would otherwise be denied, and thus possibly violating license restrictions and incurring unwarranted delivery costs.

Unauthorized access to metadata could also enable a malicious metadata client to continuously issue metadata requests in order to overload a uCDN's metadata server(s).

Unauthorized access to metadata could further result in leakage of private information. A malicious metadata client could request metadata in order to gain access to origin servers, as well as information pertaining to content restrictions.

An implementation of the CDNI metadata interface MUST use mutual authentication and message authentication codes to prevent unauthorized access to and undetected modification of metadata (see Section 8.3).

8.2. Confidentiality and Privacy

Unauthorized viewing of metadata could result in leakage of private information. Content provider origin and policy information is conveyed through the CDNI metadata interface. A third party could intercept metadata transactions in order to gain access to origin servers, as well as information pertaining to content restrictions and usage patterns.

Note: The distribution of metadata by a uCDN to dCDNs could introduce privacy concerns for some content providers, e.g., dCDNs accepting content requests for a content provider's content might be able to obtain additional information and usage patterns relating to the users of a content provider's services. Content providers with concerns about divulging information to dCDNs can instruct their uCDN partners not to use CDNI when delivering their content.

An implementation of the CDNI metadata interface MUST use strong encryption to prevent unauthorized interception or monitoring of metadata (see Section 8.3).

8.3. Securing the CDNI Metadata interface

An implementation of the CDNI metadata interface MUST support TLS transport as per [RFC2818] and [RFC7230].

TLS MUST be used by the server-side (dCDN) and the client-side (uCDN) of the CDNI metadata interface, including authentication of the remote end, unless alternate methods are used for ensuring the security of the information in the CDNI metadata interface requests and responses (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

The use of TLS for transport of the CDNI metadata interface messages allows:

- The dCDN and uCDN to authenticate each other.

and, once they have mutually authenticated each other, it allows:

- The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI metadata requests and responses from an authorized CDN);
- CDNI metadata interface requests and responses to be transmitted with confidentiality; and
- The integrity of the CDNI metadata interface requests and responses to be protected during the exchange.

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

9. Acknowledgements

The authors would like to thank David Ferguson, Francois Le Faucheur, Jan Seedorf and Matt Miller for their valuable comments and input to this document.

10. Contributing Authors

[RFC Editor Note: Please move the contents of this section to the Authors' Addresses section prior to publication as an RFC.]

Grant Watson
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: gwatson@velocix.com

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose, 95134
USA

Email: kleung@cisco.com

11. References

11.1. Normative References

[ISO3166-1] The International Organization for Standardization, "Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes", ISO 3166-1:2013, 2013.

[POSIX] Institute of Electrical and Electronics Engineers, "Information Technology Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language]", IEEE P1003.1, 1990.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, DOI 10.17487/RFC3490, March 2003, <http://www.rfc-editor.org/info/rfc3490>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc4291>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://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, <http://www.rfc-editor.org/info/rfc6707>.

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

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

11.2. Informative References

[I-D.ietf-cdni-control-triggers]
Murray, R. and B. Niven-Jenkins, "CDNI Control Interface / Triggers", draft-ietf-cdni-control-triggers-15 (work in progress), May 2016.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-18 (work in progress), April 2016.

[I-D.ietf-cdni-uri-signing]
Leung, K., Faucheur, F., Brandenburg, R., Downey, B., and M. Fisher, "URI Signing for CDN Interconnection (CDNI)", draft-ietf-cdni-uri-signing-09 (work in progress), June 2016.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC6793] Vohra, Q. and E. Chen, "BGP Support for Four-Octet Autonomous System (AS) Number Space", RFC 6793, DOI 10.17487/RFC6793, December 2012, <http://www.rfc-editor.org/info/rfc6793>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc7337>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

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

Authors' Addresses

Ben Niven-Jenkins
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: ben@velocix.com

Rob Murray
Velocix (Alcatel-Lucent)
3 Ely Road
Milton, Cambridge CB24 6AA
UK

Email: rmurray@velocix.com

Matt Caulfield
Cisco Systems
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Phone: +1 978 936 9307
Email: mcaulfie@cisco.com

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 17, 2016

B. Niven-Jenkins, Ed.
Nokia
R. van Brandenburg, Ed.
TNO

April 15, 2016

Request Routing Redirection interface for CDN Interconnection
draft-ietf-cdni-redirection-18

Abstract

The Request Routing Interface comprises (1) the asynchronous advertisement of footprint and capabilities by a downstream Content Delivery Network (CDN) that allows an upstream CDN to decide whether to redirect particular user requests to that downstream CDN; and (2) the synchronous operation of an upstream CDN requesting whether a downstream CDN is prepared to accept a user request and of a downstream CDN responding with how to actually redirect the user request. This document describes an interface for the latter part, i.e., the CDNI Request Routing Redirection interface.

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 http://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 October 17, 2016.

Table of Contents

1. Introduction

A Content Delivery Network (CDN) is a system built on an existing IP network which is used for large scale content delivery, via prefetching or dynamically caching content on its distributed surrogates (caching servers). [RFC6707] describes the problem area of interconnecting CDNs.

The CDNI Request Routing interface outlined in [RFC7336] comprises of:

1. The asynchronous advertisement of footprint and capabilities by a downstream CDN (dCDN) that allows an upstream CDN (uCDN) to decide whether to redirect particular user requests to that dCDN.

2. The synchronous operation of a uCDN requesting whether a dCDN is prepared to accept a user request and of a dCDN responding with how to actually redirect the user request.

This document describes an interface for the latter part, i.e., the CDNI Request Routing Redirection interface (RI).

2. Terminology

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 [RFC2119].

This document reuses the terminology defined in [RFC6707].

The following additional terms are introduced by this document:

Application Level Redirection: The act of using an application specific redirection mechanism for the request routing process of a CDN. The Redirection Target (RT) is the result of the routing decision of a CDN at the time it receives a content request via an application specific protocol response. Examples of an application level redirection are HTTP 302 Redirection and RTMP 302 Redirection.

DNS Redirection: The act of using DNS name resolution for the request routing process of a CDN. In DNS Redirection, the DNS name server of the CDN makes the routing decision based on a local policy and selects one or more Redirection Targets (RTs) and redirects the user agent to the RT(s) by returning the details of the RT(s) in response to the DNS query request from the user agent’s DNS resolver.

HTTP Redirection: The act of using an HTTP redirection response for the request routing process of a CDN. The Redirection Target (RT) is the result of the routing decision of a CDN at the time it receives a content request via HTTP. HTTP Redirection is a particular case of Application Level Redirection.

Redirection Target (RT): A Redirection Target is the endpoint to which the user agent is redirected. In CDNI, a RT may point to a

number of different components, some examples include a surrogate in the same CDN as the request router, a request router in a dCDN or a surrogate in a dCDN, etc.

3. Interface function and operation overview

The main function of the CDNI Redirection interface (RI) is to allow the request routing systems in interconnected CDNs to communicate to facilitate the redirection of User Agent requests between interconnected CDNs.

The detailed requirements for the Redirection interface and their relative priorities are described in section 5 of [RFC7337].

The User Agent will make a request to a request router in the uCDN using one of either DNS or HTTP. The RI is used between the uCDN and one or more dCDNs. The dCDN's RI response may contain a Redirection Target with a type that is compatible with the protocol used between User Agent and uCDN request router. The dCDN has control over the Redirection Target it provides. Depending on the returned Redirection Target, the User Agent's request may be redirected to:

- The final Surrogate, which may be in the dCDN that returned the RI response to the uCDN, or another CDN (if the dCDN delegates the delivery to another CDN); or

- A request router (in the dCDN or another CDN), which may use a different redirection protocol (DNS or HTTP) than the one included in the RI request.

The Redirection interface operates between the request routing systems of a pair of interconnected CDNs. To enable communication over the Redirection interface, the uCDN needs to know the URI (end point) in the dCDN to send CDNI request routing queries.

The Redirection interface URI may be statically pre-configured, dynamically discovered via the CDNI Control interface, or discovered via other means. However, such discovery mechanisms are not specified in this document, as they are considered out of the scope of the Redirection interface specification.

The Redirection interface is only relevant in the case of Recursive Request Redirection, as Iterative Request Redirection does not invoke any interaction over the Redirection interface between interconnected CDNs. Therefore, the scope of this document is limited to Recursive Request Redirection.

In the case of Recursive Request Redirection, in order to perform redirection of a request received from a User Agent, the uCDN queries the dCDN so that the dCDN can select and provide a Redirection Target. In cases where a uCDN has a choice of dCDNs it is up to the uCDN to decide (for example, via configured policies) which dCDN(s) to query and in which order to query them. A number of strategies are possible including selecting a preferred dCDN based on local policy, possibly falling back to querying an alternative dCDN(s) if the first dCDN does not return a Redirection Target or otherwise rejects the uCDN's RI request. A more complex strategy could be to query multiple dCDNs in parallel before selecting one and using the Redirection Target provided by that dCDN.

The uCDN->User Agent redirection protocols addressed in this draft are: DNS redirection and HTTP redirection. Other types of application level redirection will not be discussed further in this document. However, the Redirection interface is designed to be extensible and could be extended to support additional application level redirection protocols.

For both DNS & HTTP redirection, either HTTP or HTTPS could be used to connect to the Redirection Target. When HTTPS is used to connect to the uCDN, if the uCDN uses DNS redirection to identify the RT to the User Agent, then the new target domain name may not match the domain in the URL dereferenced to reach the uCDN; without operational precautions, and in the absence of DNSSEC, this can make a legitimate redirection look like a DNS-based attack to a User Agent and trigger security warnings. When DNS-based redirection with HTTPS is used, this specification assumes that any RT can complete the necessary TLS handshake with the User Agent. Any operational mechanisms this requires, e.g., private key distribution to surrogates and request routers in dCDNs, are outside the scope of this document.

This document also defines an RI loop prevention and detection mechanism as part of the Redirection interface.

4. HTTP based interface for the Redirection Interface

This document defines a simple interface for the Redirection interface based on HTTP [RFC7230], where the attributes of a User Agent's requests are encapsulated along with any other data that can aid the dCDN in processing the requests. The RI response encapsulates the attributes of the RT(s) that the uCDN should return to the User Agent (if it decides to utilize the dCDN for delivery) along with the policy for how the response can be reused. The examples of RI requests and responses below do not contain a complete set of HTTP headers for brevity; only the pertinent HTTP headers are shown.

The RI between the uCDN and dCDN uses the same HTTP interface to encapsulate the attributes of both DNS and HTTP requests received from User Agents, although the contents of the RI requests/responses contain data specific to either DNS or HTTP redirection.

This approach has been chosen because it enables CDN operators to only have to deploy a single interface for the RI between their CDNs, regardless of the User Agent redirection method. In this way, from an operational point of view there is only one interface to monitor, manage, develop troubleshooting tools for, etc.

In addition, having a single RI where the attributes of the User Agent's DNS or HTTP request are encapsulated along with the other data required for the dCDN to make a request routing decision, avoids having to try to encapsulate or proxy DNS/HTTP/RTMP/etc requests and find ways to somehow embed the additional CDNI Request Routing Redirection interface properties/data within those End User DNS/HTTP/RTMP/etc requests.

Finally, the RI is easily extendable to support other User Agent request redirection methods (e.g., RTMP 302 redirection) by defining additional protocol specific keys for RI requests and responses along with a specification how to process them.

The generic Recursive Request Redirection message flow between Request Routing systems in a pair of interconnected CDNs is as follows:

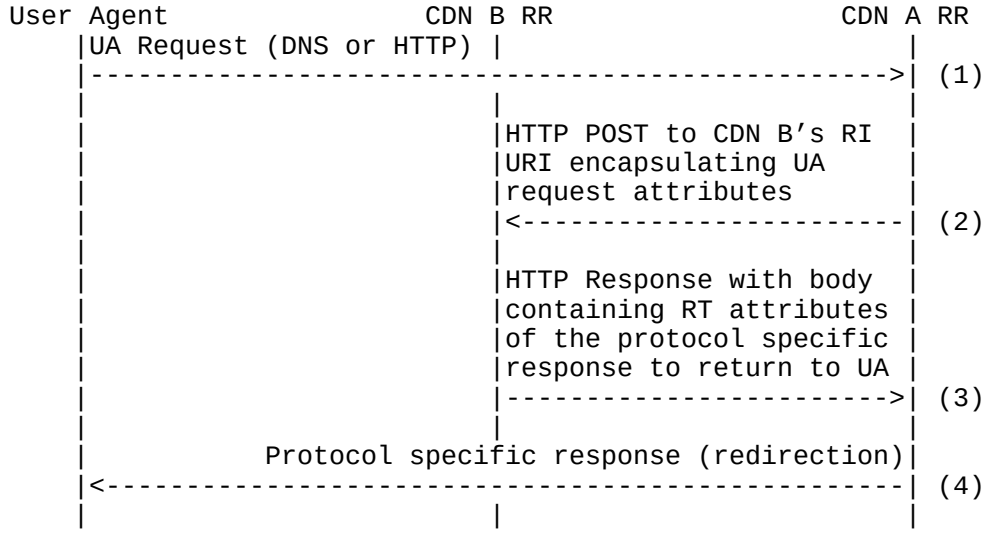


Figure 1: Generic Recursive Request Redirection message flow

1. The User Agent sends its (DNS or HTTP) request to CDN A. The Request Routing System of CDN A processes the request and, through local policy, recognizes that the request is best served by another CDN, specifically CDN B (or that CDN B may be one of a number of candidate dCDNs it could use).

2. The Request Routing System of CDN A sends an HTTP POST to CDN B's RI URI containing the attributes of the User Agent's request.

3. The Request Routing System of CDN B processes the RI request and assuming the request is well formed, responds with an HTTP "200" response with a message body containing the RT(s) to return to the User Agent as well as parameters that indicate the properties of the response (cacheability and scope).

4. The Request Routing System of CDN A sends a protocol specific response (containing the returned attributes) to the User Agent, so that the User Agent's request will be redirected to the RT(s) returned by CDN B.

4.1. Information passed in RI requests & responses

The information passed in RI requests splits into two basic categories:

1. The attributes of the User Agent's request to the uCDN.

2. Properties/parameters that the uCDN can use to control the dCDN's response or that can help the dCDN make its decision.

To assist the routing decision of a dCDN, the uCDN SHOULD convey as much information as possible to the dCDN, for example, the URI of the requested content and the User Agent's IP address or subnet, when those are known by the uCDN Request Routing system.

In order for the dCDN to determine whether it is capable of delivering any requested content, it requires CDNI metadata related to the content the User Agent is requesting. That metadata will describe the content and any policies associated with it. It is expected that the RI request contains sufficient information for the Request Router in the dCDN to be able to retrieve the required CDNI Metadata via the CDNI Metadata interface.

The information passed in RI responses splits into two basic categories:

1. The attributes of the RT to return to the User Agent in the DNS response or HTTP response.

2. Parameters/policies that indicate the properties of the response, such as, whether it is cacheable, the scope of the response, etc.

In addition to details of how to redirect the User Agent, the dCDN may wish to return additional policy information to the uCDN to it with future RI requests. For example, the dCDN may wish to return a policy that expresses "this response can be reused without requiring an RI request for 60 seconds provided the User Agent's IP address is in the range 198.51.100.0 - 198.51.100.255".

These additional policies split into two basic categories:

- Cacheability information signaled via the HTTP response headers of the RI response (to reduce the number of subsequent RI requests the uCDN needs to make).

- The scope of a cacheable response signaled in the HTTP response body of the RI response, for example, whether the response applies to a wider range of IP addresses than what was included in the RI request.

The cacheability of the response is indicated using the standard HTTP Cache-Control mechanisms.

4.2. JSON encoding of RI requests & responses

The body of RI requests and responses is a JSON object [RFC7159] that MUST conform to [RFC7493] containing a dictionary of key:value pairs. Senders MUST encode all (top level object and sub-object) keys specified in this document in lowercase. Receivers MUST ignore any keys that are unknown or invalid.

The following top level keys are defined along with whether they are applicable to RI requests, RI responses or both:

Key	Request/Response	Description
dns	Both	The attributes of the UA's DNS request or the attributes of the RT(s) to return in a DNS response.
http	Both	The attributes of the UA's HTTP request or the attributes of the RT to return in a HTTP response.
scope	Response	The scope of the response (if it is cacheable). For example, whether the response applies to a wider range of IP addresses than what was included in the RI request.
error	Response	Additional details if the response is an error response.
cdn-path	Both	A List of Strings. Contains a list of the CDN Provider IDs of previous CDNs that have participated in the request routing for the associated User Agent request. On RI requests it contains the list of previous CDNs that this RI request has passed through. On RI responses it contains the list of CDNs that were involved in obtaining the final redirection included in the RI response.
max-hops	Request	Integer specifying the maximum number of hops (CDN Provider IDs) this request is allowed to be propagated along. This allows the uCDN to coarsely constrain the latency of the request routing chain.

Top-Level keys in RI requests/responses

A single request or response MUST contain only one of the dns or http keys. Requests MUST contain a cdn-path key and responses MAY contain a cdn-path key. If the max-hops key is not present then there is no limit on the number of CDN hops that the RI request can be propagated along. If the first uCDN does not wish the RI request to be propagated beyond the dCDN it is making the request to, then the uCDN MUST set max-hops to 1.

The cdn-path MAY be reflected back in RI responses, although doing so could expose information to the uCDN that a dCDN may not wish to expose (for example, the existence of business relationships between a dCDN and other CDNs).

If the cdn-path is reflected back in the RI response it MUST contain the value of cdn-path received in the associated RI request with the final dCDN's CDN Provider ID appended. Transit CDNs MAY remove the cdn-path from RI responses but MUST NOT modify the cdn-path in other ways.

The presence of an error key within a response that also contains either a dns or http key does not automatically indicate that the RI request was unsuccessful as the error key MAY be used for communicating additional (e.g., debugging) information. When a response contains an error key as well as either a dns or http key, the error-code SHOULD be 1xx (e.g., 100). See Section 4.7 for more details of encoding error information in RI responses.

All implementations that support IPv4 addresses MUST support the encoding specified by the 'IPv4address' rule in Section 3.2.2 of [RFC3986]. Likewise, implementations that support IPv6 addresses MUST support all IPv6 address formats specified in [RFC4291]. Server implementations SHOULD use IPv6 address formats specified in [RFC5952].

4.3. MIME Media Types used by the RI interface

RI requests MUST use a MIME Media Type of application/cdni as specified in [RFC7736], with the Payload Type (ptype) parameter set to 'redirection-request'.

RI responses MUST use a MIME Media Type of application/cdni as specified in [RFC7736], with the Payload Type (ptype) parameter set to 'redirection-response'.

4.4. DNS redirection

The following sections provide detailed descriptions of the information that should be passed in RI requests and responses for DNS redirection.

4.4.1. DNS Redirection requests

For DNS based redirection the uCDN needs to pass the following information to the dCDN in the RI request:

- The IP address of the DNS resolver that made the DNS request to the uCDN.
- The type of DNS query made (usually either A or AAAA).
- The class of DNS query made (usually IN).
- The fully qualified domain name for which DNS redirection is being requested.
- The IP address or prefix of the User Agent (if known to the uCDN).

The information above is encoded as a set of key:value pairs within the dns dictionary as follows:

Key	Value	Mandatory	Description
resolver-ip	String	Yes	The IP address of the UA's DNS resolver.
qtype	String	Yes	The type of DNS query made by the UA's DNS resolvers in uppercase (A, AAAA, etc.).
qclass	String	Yes	The class of DNS query made in uppercase (IN, etc.).
qname	String	Yes	The fully qualified domain name being queried.
c-subnet	String	No	The IP address (or prefix) of the UA in CIDR format.
dns-only	Boolean	No	If True then dCDN MUST only use DNS redirection and MUST include RTs to one or more surrogates in any successful RI response. CDNs MUST include the dns-only property set to True on any cascaded RI requests. Defaults to False.

An RI request for DNS-based redirection MUST include a dns dictionary. This dns dictionary MUST contain the following keys: resolver-ip, qtype, qclass, qname and the value of each MUST be the value of the appropriate part of the User Agent's DNS query/request.

An example RI request (uCDN->dCDN) for DNS based redirection:

```
POST /dcdn/ri HTTP/1.1
Host: rr1.dcdn.example.net
Content-Type: application/cdni; ptype=redirection-request
Accept: application/cdni; ptype=redirection-response

{
  "dns" : {
    "resolver-ip" : "192.0.2.1",
    "c-subnet" : "198.51.100.0/24",
    "qtype" : "A",
    "qclass" : "IN",
    "qname" : "www.example.com"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}
```

4.4.2. DNS Redirection responses

For a successful DNS based redirection, the dCDN needs to return one of the following to the uCDN in the RI response:

- The IP address(es) of (or the CNAME of) RTs that are dCDN surrogates (if the dCDN is performing DNS based redirection directly to a surrogate); or

- The IP address(es) of (or the CNAME of) RTs that are Request Routers (if the dCDN will perform request redirection itself). A dCDN MUST NOT return a RT which is a Request Router if the dns-only key is set to True in the RI request.

The information above is encoded as a set of key:value pairs within the dns dictionary as follows:

Key	Value	Mandatory	Description
rcode	Integer	Yes	DNS response code (see [RFC6895]).
name	String	Yes	The fully qualified domain name the response relates to.
a	List of String	No	Set of IPv4 Addresses of RT(s).
aaaa	List of String	No	Set of IPv6 Addresses of RT(s).
cname	List of String	No	Set of fully qualified domain names of RT(s).
ttl	Integer	No	TTL in seconds of DNS response. Default is 0.

A successful RI response for DNS-based redirection MUST include a dns dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for DNS-based redirection MUST include an error dictionary. If a dns dictionary is included in the RI response, it MUST include the rcode and name keys and it MUST include at least one of the following keys: a, aaaa, cname. The dns dictionary MAY include both 'a' and 'aaaa' keys. If the dns dictionary contains a cname key it MUST NOT contain either an a or aaaa key.

An example of a successful RI response (dCDN->uCDN) for DNS based redirection with both a and aaaa keys is listed below :

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response

{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201", "203.0.113.202"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
}
```

A further example of a successful RI response (dCDN->uCDN) for DNS based redirection is listed below, in this case with a cname key containing the FQDN of the RT.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response

{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "cname" : ["rr1.dcdn.example"],
    "ttl" : 20
  }
}
```

4.5. HTTP Redirection

The following sections provide detailed descriptions of the information that should be passed in RI requests and responses for HTTP redirection.

The dictionary keys used in HTTP Redirection requests and responses use the following conventions for their prefixes:

- c- is prefixed to keys for information related to the Client (User Agent).
- cs- is prefixed to keys for information passed by the Client (User Agent) to the Server (uCDN).
- sc- is prefixed to keys for information to be passed by the Server (uCDN) to the Client (User Agent).

4.5.1. HTTP Redirection requests

For HTTP-based redirection the uCDN needs to pass the following information to the dCDN in the RI request:

- The IP address of the User Agent.
- The URI requested by the User Agent.
- The HTTP method requested by the User Agent
- The HTTP version number requested by the User Agent.

The uCDN may also decide to pass the presence and value of particular HTTP headers included in the User Agent request to the dCDN.

The information above is encoded as a set of key:value pairs within the http dictionary as follows:

Key	Value	Mandatory	Description
c-ip	String	Yes	The IP address of the UA.
cs-uri	String	Yes	The Effective Request URI [RFC7230] requested by the UA.
cs-method	String	Yes	The method part of the request-line as defined in Section 3.1.1 of [RFC7230].
cs-version	String	Yes	The HTTP-version part of the request-line as defined in Section 3.1.1 of [RFC7230].
cs-(<headername>)	String	No	The field-value of the HTTP header field named <HeaderName> as a string, for example, cs-(cookie) would contain the value of the HTTP Cookie header from the UA request.

An RI request for HTTP-based redirection MUST include an http dictionary. This http dictionary MUST contain the following keys: c-ip, cs-method, cs-version and cs-uri and the value of each MUST be the value of the appropriate part of the User Agent’s HTTP request.

The http dictionary of an RI request MUST contain a maximum of one cs-(<headername>) key for each unique header field-name (HTTP header field). <headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

In the case where the User Agent request includes multiple HTTP header fields with the same field-name, it is RECOMMENDED that the uCDN combines these different HTTP headers into a single value according to Section 3.2.2 of [RFC7230]. However, because of the plurality of already defined HTTP header fields, and inconsistency of some of these header fields concerning the combination mechanism defined in RFC 7230, the uCDN MAY have to deviate from using the combination mechanism where appropriate. For example, it might only

send the contents of the first occurrence of the HTTP Headers instead.

An example RI request (uCDN->dCDN) for HTTP based redirection:

```
POST /dcdn/rrri HTTP/1.1
Host: rr1.dcdn.example.net
Content-Type: application/cdni; ptype=redirection-request
Accept: application/cdni; ptype=redirection-response

{
  "http": {
    "c-ip": "198.51.100.1",
    "cs-uri": "http://www.example.com",
    "cs-version": "HTTP/1.1",
    "cs-method": "GET"
  },
  "cdn-path": ["AS64496:0"],
  "max-hops": 3
}
```

4.5.2. HTTP Redirection responses

For a successful HTTP based redirection, the dCDN needs to return one of the following to the uCDN in the RI response:

- A URI pointing to an RT that is the selected dCDN surrogate(s) (if the dCDN is performing HTTP based redirection directly to a surrogate); or

- A URI pointing to an RT that is a Request Router (if the dCDN will perform request redirection itself).

The information above is encoded as a set of key:value pairs within the http dictionary as follows:

Key	Value	Mandatory	Description
sc-status	Integer	Yes	The status-code part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA (usually set to 302).
sc-version	String	Yes	The HTTP-version part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
sc-reason	String	Yes	The reason-phrase part of the status-line as defined in Section 3.1.2 of [RFC7230] to return to the UA.
cs-uri	String	Yes	The URI requested by the UA/client.
sc-(location)	String	Yes	The contents of the Location header to return to the UA (i.e., a URI pointing to the RT(s)).
sc-(<headername>)	String	No	The field-value of the HTTP header field named <HeaderName> to return to the UA. For example, sc-(expires) would contain the value of the HTTP Expires header.

Note: The sc-(location) key in the table above is an example of sc-(<headername>) that has been called out separately as its presence is mandatory in RI responses.

A successful RI response for HTTP-based redirection MUST include an http dictionary and MAY include an error dictionary (see Section 4.7). An unsuccessful RI response for HTTP-based redirection MUST include an error dictionary. If an http dictionary is included in the RI response, it MUST include at least the following keys: sc-status, sc-version, sc-reason, cs-uri and sc-(location).

The http dictionary of an RI response MUST contain a maximum of one sc-(<headername>) key for each unique header field-name (HTTP header field). <headername> MUST be identical to the equivalent HTTP header field-name encoded in all lowercase.

The uCDN MAY decide to not return, override or alter any or all of the HTTP headers defined by sc-(<headername>) keys before sending the HTTP response to the UA. It should be noted that in some cases, sending the HTTP Headers indicated by the dCDN transparently on to the UA might result in, for the uCDN, undesired behaviour. As an example, the dCDN might include sc-(cache-control), sc-(last-modified) and sc-(expires) keys in the http dictionary, through which the dCDN may try to influence the cacheability of the response by the UA. If the uCDN would pass these HTTP headers on to the UA, this could mean that further requests from the uCDN would go directly to the dCDN, bypassing the uCDN and any logging it may perform on incoming requests. The uCDN is therefore recommended to carefully consider which HTTP headers to pass on, and which to either override or not pass on at all.

An example of a successful RI response (dCDN->uCDN) for HTTP based redirection:

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response

{
  "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://sur1.dcdn.example/ucdn/example.com",
  }
}
```

4.6. Cacheability and scope of responses

RI responses may be cacheable. As long as a cached RI response is not stale according to standard HTTP Cache-Control or other applicable mechanisms, it may be reused by the uCDN in response to User Agent requests without sending another RI request to the dCDN.

An RI response MUST NOT be reused unless the request from the User Agent would generate an identical RI request to the dCDN as the one that resulted in the cached RI response (except for the c-ip field

provided that the User Agent's c-ip is covered by the scope in the original RI response, as elaborated upon below).

Additionally, although RI requests only encode a single User Agent request to be redirected there may be cases where a dCDN wishes to indicate to the uCDN that the RI response can be reused for other User Agent requests without the uCDN having to make another request via the RI. For example, a dCDN may know that it will always select the same Surrogates for a given set of User Agent IP addresses and in order to reduce request volume across the RI or to remove the additional latency associated with an RI request, the dCDN may wish to indicate that set of User Agent IP addresses to the uCDN in the initial RI response. This is achieved by including an optional scope dictionary in the RI response.

Scope is encoded as a set of key:value pairs within the scope dictionary as follows:

Key	Value	Mandatory	Description
iprange	List of String	No	A List of IP subnets in CIDR notation that this RI response can be reused for, provided the RI response is still considered fresh.

If a uCDN has multiple cached responses with overlapping scopes and a UA request comes in for which the User Agent's IP matches with the IP subnets in multiple of these cached responses, the uCDN SHOULD use the most recent cached response when determining the approriate RI response to use.

The following is an example of a DNS redirection response from Section 4.4.2 that is cacheable by the uCDN for 30 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: public, max-age=30

{
  "dns" : {
    "rcode" : 0,
    "name" : "www.example.com",
    "a" : ["203.0.113.200", "203.0.113.201"],
    "aaaa" : ["2001:DB8::C8", "2001:DB8::C9"],
    "ttl" : 60
  }
  "scope" : {
    "iprange" : ["198.51.100.0/24"]
  }
}
```

Example of HTTP redirection response from Section 4.5.2 that is cacheable by the uCDN for 60 seconds and can be returned to any User Agent with an IPv4 address in 198.51.100.0/24.

Note: The response to the UA is only valid for 30 seconds, whereas the uCDN can cache the RI response for 60 seconds.

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: public, max-age=60

{
  "http": {
    "sc-status": 302,
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://sur1.dcdn.example/ucdn/example.com",
    "sc-(cache-control)" : "public, max-age=30"
  }
  "scope" : {
    "iprange" : ["198.51.100.0/24"]
  }
}
```

4.7. Error responses

From a uCDN perspective, there are two types of errors that can be the result of the transmission of an RI request to a dCDN:

1. An HTTP protocol error signaled via an HTTP status code, indicating a problem with the reception or parsing of the RI request or the generation of the RI response by the dCDN, and

2. An RI-level error specified in an RI response message

This section deals with the latter type. The former type is outside the scope of this document.

There are numerous reasons for a dCDN to be unable to return an affirmative RI response to a uCDN. Reasons may include both dCDN internal issues such as capacity problems, as well as reasons outside the influence of the dCDN, such as a malformed RI request. To aid with diagnosing the cause of errors, RI responses SHOULD include an error dictionary to provide additional information to the uCDN as to the reason/cause of the error. The intention behind the error dictionary is to aid with either manual or automatic diagnosis of issues. The resolution of such issues is outside the scope of this document; this document does not specify any consequent actions a uCDN should take upon receiving a particular error code.

Error information (if present) is encoded as a set of key:value pairs within a JSON-encoded error dictionary as follows:

Key	Value	Mandatory	Description
error-code	Integer	Yes	A three-digit numeric code defined by the server to indicate the error(s) that occurred.
reason	String	No	A string providing further information related to the error.

The first digit of the error-code defines the class of error. There are 5 classes of error distinguished by the first digit of the error-code:

1xx: Informational (no error): The response should not be considered an error by the uCDN, which may proceed by redirecting the UA according to the values in the RI response. The error code and accompanying description may be used for informational purposes, e.g., for logging.

2xx: Reserved.

3xx: Reserved.

4xx: uCDN error: The dCDN can not or will not process the request due to something that is perceived to be a uCDN error, for example, the RI request could not be parsed succesfully by the dCDN. The last two-digits may be used to more specifically indicate the source of the problem.

5xx: dCDN error: Indicates that the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason, for example, the dCDN was able to parse the RI request but encountered an error for some reason. Examples include the dCDN not being able to retrieve the associated metadata or the dCDN being out of capacity.

The following error codes are defined and maintained by IANA (see Section 6):

Error codes with a "Reason" of "<reason>" do not have a defined value for their 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically.

Code	Reason	Description
100	<reason> (see Description)	Generic informational error-code meant for carrying a human-readable string
400	<reason> (see Description)	Generic error-code for uCDN errors where the dCDN can not or will not process the request due to something that is perceived to be a uCDN error. The reason field may be used to provide more details about the source of the error.
500	<reason> (see Description)	Generic error-code for dCDN errors where the dCDN is aware that it has erred or is incapable of satisfying the RI request for some reason. The reason field may be used to provide more details about the source of the error.
501	Unable to retrieve metadata	The dCDN is unable to retrieve the metadata associated with the content requested by the UA. This may indicate a configuration error or the content requested by the UA not existing.
502	Loop detected	The dCDN detected a redirection loop (see Section 4.8).
503	Maximum hops exceeded	The dCDN detected the maximum number of redirection hops exceeding max-hops (see Section 4.8).
504	Out of capacity	The dCDN does not currently have sufficient capacity to handle the UA request.
505	Delivery protocol not supported	The dCDN does not support the (set of) delivery protocols indicated in the CDNI Metadata of the content requested content by the UA.
506	Redirection protocol not supported	The dCDN does not support the requested redirection protocol. This error-code is also used when the RI request has the dns-only flag set to True and the dCDN is not support or is not prepared to return a RT of a surrogate directly.

Table 1

The following is an example of an unsuccessful RI response (dCDN->uCDN) for a DNS based User Agent request:

```
HTTP/1.1 500 Internal Server Error
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache

{
  "error" : {
    "error-code" : 504,
    "description" : "Out of capacity"
  }
}
```

The following is an example of a successful RI response (dCDN->uCDN) for a HTTP based User Agent request containing an error dictionary for informational purposes:

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2012 18:41:38 GMT
Content-Type: application/cdni; ptype=redirection-response
Cache-Control: private, no-cache

{
   "http": {
    "sc-status": 302,
    "sc-version": "HTTP/1.1",
    "sc-reason": "Found",
    "cs-uri": "http://www.example.com"
    "sc-(location)":
      "http://sur1.dcdn.example/ucdn/example.com",
   },
   "error" : {
    "error-code" : 100,
    "description" :
      "This is a human-readable message meant for debugging purposes"
  }
}
```

4.8. Loop detection & prevention

In order to prevent and detect RI request loops, each CDN MUST insert its CDN Provider ID into the cdn-path key of every RI request it originates or cascades. When receiving RI requests a dCDN MUST check the cdn-path and reject any RI requests which already contain the dCDN’s Provider ID in the cdn-path. Transit CDNs MUST NOT propagate to any downstream CDNs if the number of CDN Provider IDs in cdn-path (before adding its own Provider ID) is equal to or greater than max-hops.

The CDN Provider ID uniquely identifies each CDN provider during the course of request routing redirection. It consists of the characters AS followed by the CDN Provider's AS number, then a colon (':') and an additional qualifier that is used to guarantee uniqueness in case a particular AS has multiple independent CDNs deployed. For example, "AS64496:0".

If a dCDN receives an RI request whose cdn-path already contains that dCDN's Provider ID the dCDN MUST send an RI error response which SHOULD include an error code of 502.

If a dCDN receives an RI request where the number of CDN Provider IDs in cdn-path is greater than max-hops, the dCDN MUST send an RI error response which SHOULD include an error code of 503.

It should be noted that the loop detection & prevention mechanisms described above only cover preventing and detecting loops within the RI itself. Besides loops within the RI itself, there is also the possibility of loops in the data plane, for example, if the IP address(es) or URI(s) returned in RI responses do not resolve directly to a surrogate in the final dCDN there is the possibility that a User Agent may be continuously redirected through a loop of CDNs. The specification of solutions to address data plane request redirection loops between CDNs is outside of the scope of this document.

5. Security Considerations

Information passed over the RI could be considered personal or sensitive, for example, RI requests contain parts of a User Agent's original request and RI responses reveal information about the dCDN's policy for which surrogates should serve which content/user locations.

The RI interface also provides a mechanism whereby a uCDN could probe a dCDN and infer the dCDN's edge topology by making repeated RI requests for different content and/or UA IP addresses and correlating the responses from the dCDN. Additionally the ability for a dCDN to indicate that an RI response applies more widely than the original request (via the scope dictionary) may significantly reduce the number of RI requests required to probe and infer the dCDN's edge topology.

The same information could be obtained in the absence of the RI interface, but it could be more difficult to gather as it would require a distributed set of machines with a range of different IP addresses each making requests directly to the dCDN. However, the RI facilitates easier collection of such information as it enables a

single client to query the dCDN for a redirection/surrogate selection on behalf of any UA IP address.

5.1. Authentication, Authorization, Confidentiality, Integrity Protection

An implementation of the CDNI Redirection interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI Redirection interface messages allows:

- The dCDN and uCDN to authenticate each other

and, once they have mutually authenticated each other, it allows:

- The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI Redirection messages to/from an authorized CDN);
- CDNI Redirection interface messages to be transmitted with confidentiality; and
- The integrity of the CDNI Redirection interface messages to be protected during the exchange.

In an environment where any such protection is required, mutually authenticated encrypted transport MUST be used to ensure confidentiality of the redirection information, and to do so, TLS MUST be used (including authentication of the remote end) by the server-side (dCDN) and the client-side (uCDN) of the CDNI Redirection interface.

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

5.2. Privacy

Information passed over the RI ought to be considered personal and sensitive. In particular, parts of a User Agent’s original request, most notably the UA’s IP address and requested URI, are transmitted over the RI to the dCDN. The use of mutually authenticated TLS, as described in the previous section, prevents any other party than the authorized dCDN from gaining access to this information.

Regardless of whether the uCDN and dCDN use the RI, a successfull redirect from a uCDN to a dCDN will make that dCDN aware of the UA’s IP address. As such, the fact that this information is transmitted across the RI does not allow the dCDN to learn new information. On the other hand, if a uCDN uses the RI to check with multiple

candidate dCDNs, those candidates that do not end up getting redirected to, do obtain information regarding End User IP addresses and requested URIs that they would not have, had the RI not been used.

While it is technically possible to mask some information in the RI Request, such as the last bits of the UA IP address, it is important to note that this will reduce the effectivess of the RI in certain cases. CDN deployments need to strike a balance between end-user privacy and the features impacted by such masking. This balance is likely to vary from one deployment to another. As an example, when the UA and its DNS resolver is behind a Carrier-grade NAT, and the RI is used to find an appropriate delivery node behind the same NAT, the full IP address might be necessary. Another potential issue when using IP anonymization is that it is no longer possible to correlate an RI Request with a subsequent UA request.

6. IANA Considerations

6.1. CDNI Payload Type Parameter registrations

The IANA is requested to register the following two new Payload Types in the CDNI Payload Type Parameter registry for use with the application/cdni MIME media type.

[RFC Editor Note: Please replace the references to [RFCthis] below with this document's RFC number before publication.

Payload Type	Specification
redirection-request	[RFCthis]
redirection-response	[RFCthis]

6.1.1. CDNI RI Redirection Request Payload Type

Purpose: The purpose of this payload type is to distinguish RI request messages.

Interface: RI

Encoding: see Section 4.4.1 and Section 4.5.1

6.1.2. CDNI RI Redirection Response Payload Type

Purpose: The purpose of this payload type is to distinguish RI response messages.

Interface: RI

Encoding: see Section 4.4.2 and Section 4.5.2

6.2. RI Error response registry

IANA is requested to create a new "CDNI RI Error response code" subregistry within the "Content Delivery Network Interconnection (CDNI) Parameters" registry. The "CDNI RI Error response code" namespace defines the valid values for the error-code key in RI error responses. The CDNI RI Error response code MUST be a three digit integer.

Additions to the "RI Error response registry" will be made via "Specification Required" as defined in [RFC5226].

The Designated Expert will verify that new error code registrations do not duplicate existing error code definitions (in name or functionality), ensure that the new error code is in accordance with the error classes defined in section Section 4.7 of this document, prevent gratuitous additions to the namespace, and prevent any additions to the namespace that would impair the interoperability of CDNI implementations.

New registrations are required to provide the following information:

Code: A three-digit numeric error-code, in accordance with the error classes defined in section Section 4.7 of this document.

Reason: A string that provides further information related to the error that will be included in the JSON error dictionary with the 'reason'-key. Depending on the error-code semantics, the value of this field may be determined dynamically. In that case, the registration should set this value to '<reason>' and define its semantics in the description field.

Description: A brief description of the error code semantics.

Specification: An optional reference to a specification that defines in the error code in more detail.

The entries in Table 1 are registered by this document.

7. Contributors

[RFC Editor Note: Please move the contents of this section to the Authors' Addresses section prior to publication as an RFC.]

The following persons have participated as co-authors to this document:

Wang Danhua, Huawei, Email: wangdanhua@huawei.com

He Xiaoyan, Huawi, Email: hexiaoyan@huawei.com

Ge Chen, China Telecom, Email: cheng@gsta.com

Ni Wei, China Mobile, Email: niwei@chinamobile.com

Yunfei Zhang, Email:hishigh@gmail.com

Spencer Dawkins, Huawei, Email: spencer@wonderhamster.org

8. Acknowledgements

The authors would like to thank Taesang Choi, Francois le Faucheur, Matt Miller, Scott Wainner and Kevin J Ma for their valuable comments and input to this document.

9. References

9.1. 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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc4291>.

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

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

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

[RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <http://www.rfc-editor.org/info/rfc6895>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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, <http://www.rfc-editor.org/info/rfc6707>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

9.2. Informative References

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

[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, <http://www.rfc-editor.org/info/rfc7336>.

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

Authors' Addresses

Ben Niven-Jenkins (editor)
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
UK

Email: ben.niven-jenkins@nokia.com

Ray van Brandenburg (editor)
TNO
Anna van Buerenplein 1
The Hague 2595DA
the Netherlands

Phone: +31-88-866-7000
Email: ray.vanbrandenburg@tno.nl

CDNI
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2016

K. Leung
F. Le Faucheur
Cisco Systems
R. van Brandenburg
TNO
B. Downey
Verizon Labs
M. Fisher
Limelight Networks
June 28, 2016

URI Signing for CDN Interconnection (CDNI)
draft-ietf-cdni-uri-signing-09

Abstract

This document describes how the concept of URI signing supports the content access control requirements of CDNI and proposes a URI signing scheme.

The proposed URI signing method specifies the information needed to be included in the URI and the algorithm used to authorize and to validate access requests for the content referenced by the URI. 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 http://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 December 30, 2016.

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 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]. In this document, along with the CDNI Requirements [RFC7337] document and the CDNI Framework [RFC7336] the need for interconnected CDNs to be able to implement an access control mechanism that enforces the CSP's distribution policy is described.

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 Distribution 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:

* need to validate URI signed information (e.g., Expiry time, Client IP address)."

This document proposes a URI Signing scheme 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 CSP and the role of validation of 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.

1.1. Terminology

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 [RFC2119].

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

This document also uses the terminology of Keyed-Hashing for Message Authentication (HMAC) [RFC2104].

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

- URI Signature: Message digest or digital signature that is computed with an algorithm for protecting the URI.
- Full Original URI: The URI before URI Signing is applied.
- Signed URI: Any URI that contains a URI Signature.
- Target CDN URI: Embedded URI created by the CSP to direct UA towards the Upstream CDN. The Target CDN URI can be signed by the CSP and verified by the Upstream CDN.
- Redirection URI: URI created by the Upstream CDN to redirect UA towards the Downstream CDN. The Redirection URI can be signed by the Upstream CDN and verified by the Downstream CDN. In a

cascaded CDNI scenario, there can be more than one Redirection URI.

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 attributes 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). Of course, the attributes need to be added to the URI in a way that prevents a UA from changing the attributes, thereby leaving the CDN to think that the request was authorized by the CSP when in fact it wasn't. For this reason, a URI Signing mechanism includes in the URI a message digest or digital signature that allows a CDN to check the authenticity of the URI. The message digest or digital signature can be calculated based on a shared secret between the CSP and CDN or using CSP's asymmetric public/private key pair, respectively.

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 URI Signature which is generated by the CSP using the shared secret or a private key. Once the UA receives the response with the embedded URI, it sends a new HTTP request using the embedded URI to the CDN (#3). Upon receiving the request, the CDN checks to see if the Signed URI is authentic by verifying the URI signature. 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 values are confirmed to be valid, the CDN delivers the content (#4).

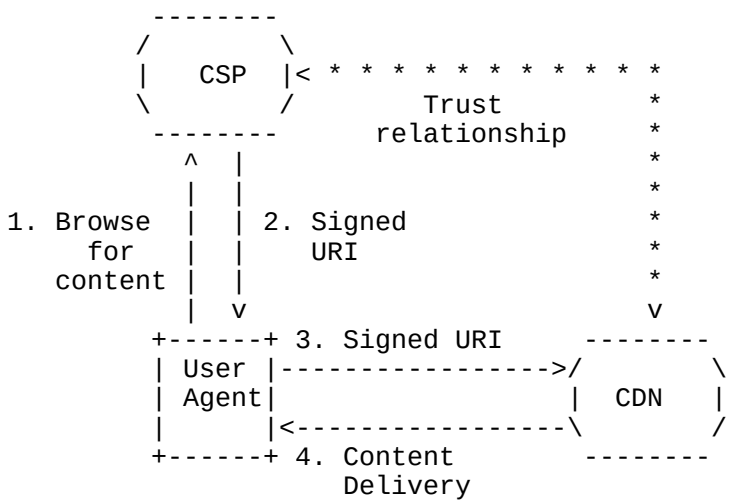
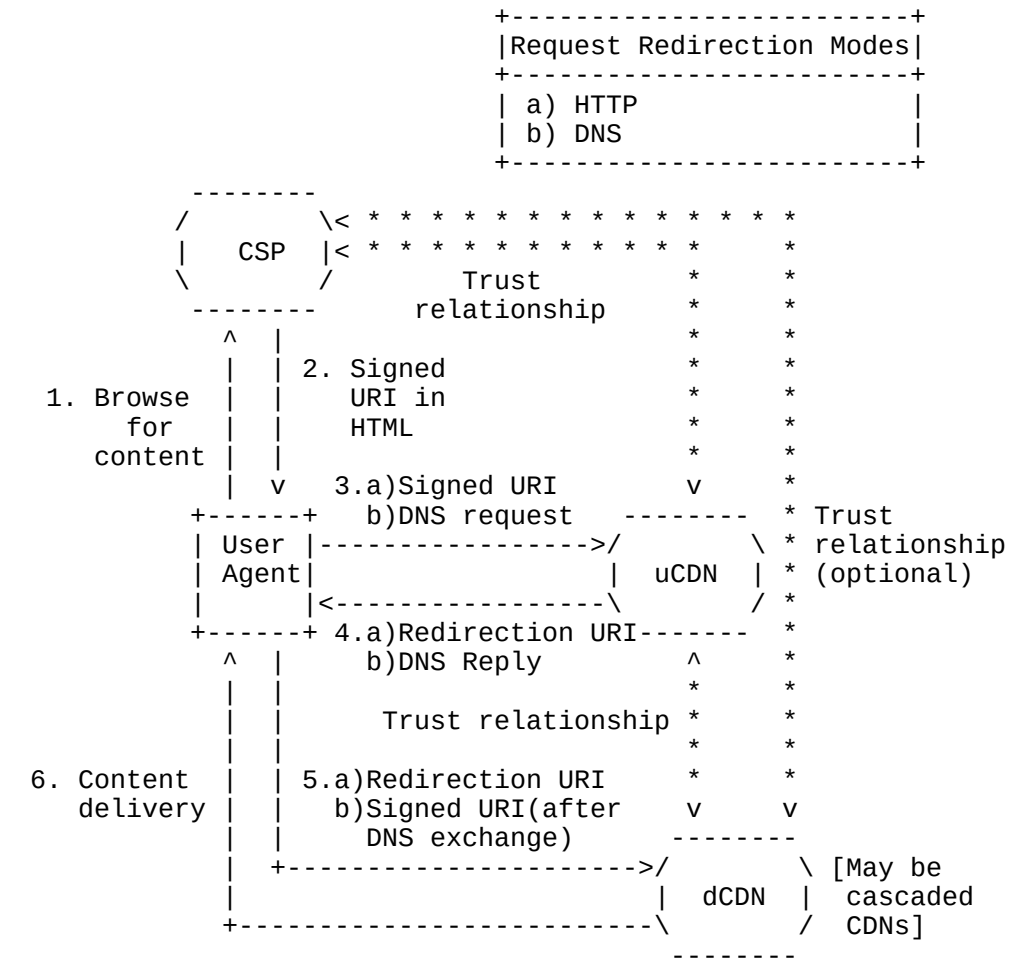


Figure 1: Figure 1: URI Signing in a CDN Environment

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 Upstream CDN and the Downstream CDN. In step #3, UA may send HTTP request or DNS request. Depending on whether HTTP-based or DNS-based request routing is used, the Upstream CDN responds by directing the UA towards the Downstream CDN using either a Redirection URI (which is a Signed URI generated by the Upstream CDN) or a DNS reply, respectively (#4). Once the UA receives the response, it sends the Redirection URI/Target CDN URI to the Downstream CDN (#5). The received URI is validated by the Downstream CDN 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 6).



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

Figure 2: URI Signing in a CDNI Environment

The trust relationships between CSP, Upstream CDN, and Downstream CDN 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 Upstream CDN. The delivery of the content may be delegated to the

Downstream CDN, which has a relationship with the Upstream CDN 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 Downstream CDN directly. In the case where the Downstream CDN does not have a trust relationship with the CSP, this means that only an asymmetric public/private key method can be used for computing the URI Signature because the CSP and Downstream CDN are not able to exchange symmetric shared secret keys. Since the CSP is unlikely to have relationships with all the Downstream CDNs that are delegated to by the Upstream CDN, the CSP may choose to allow the Authoritative CDN to redistribute the shared key to a subset of their Downstream CDNs .

For HTTP-based request routing, the Signed URI (i.e., Target CDN URI) provided by the CSP reaches the Upstream CDN. After this URI has been verified to be correct by the Upstream CDN, the Upstream CDN creates and signs a new Redirection URI to redirect the UA to the Downstream CDN. Since this new URI also has a new URI Signature, this new signature can be based around the trust relationship between the Upstream CDN and Downstream CDN, and the relationship between the Downstream CDN and CSP is not relevant. Given the fact that such a relationship between Upstream CDN and Downstream CDN always exists, both asymmetric public/private keys and symmetric shared secret keys can be used for URI Signing. Note that the signed Redirection URI MUST maintain the same, or higher, level of security as the original Signed URI.

1.4. URI Signing in a non-CDNI context

While the URI signing scheme 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 scheme 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. Signed URI Information Elements

The concept behind URI Signing is based on embedding in the Target CDN URI/Redirection URI a number of information elements that can be validated to ensure the UA has legitimate access to the content. These information elements are appended, in an encapsulated form, to the original URI.

For the purposes of the URI signing mechanism described in this document, three types of information elements may be embedded in the URI:

- Enforcement Information Elements: Information Elements that are used to enforce a distribution policy defined by the CSP. Examples of enforcement attributes are IP address of the UA and time window.

- Signature Computation Information Elements: Information Elements that are used by the CDN to verify the URI signature embedded in the received URI. In order to verify a URI Signature, the CDN requires some information elements that describe how the URI Signature was generated. Examples of Signature Computation Elements include the used HMACs hash function and/or the key identifier.

- URI Signature Information Elements: The information elements that carry the actual message digest or digital signature representing the URI signature used for checking the integrity and authenticity of the URI. A typical Signed URI will only contain one embedded URI Signature Information Element.

In addition, the this document specifies the following URI attribute:

- URI Signing Package Attribute: The URI attribute that encapsulates all the URI Signing information elements in an encoded format. Only this attribute is exposed in the Signed URI as a URI query parameter or as URL path parameter.

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.

Note that all the URI Signing information elements and the URI query attribute are mandatory to implement, but not mandatory to use.

2.1. Enforcement Information Elements

This section identifies the set of information elements that may be needed to enforce the CSP distribution policy. New information elements may be introduced in the future to extend the capabilities of the distribution policy.

In order to provide flexibility in distribution policies to be enforced, the exact subset of information elements used in the URI Signature of a given request is a deployment decision. The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to enforce the distribution policy:

- Expiry Time (ET) [optional] - Time when the Signed URI expires. This is represented as an integer denoting the number of seconds since midnight 1/1/1970 UTC (i.e., UNIX epoch). The request is rejected if the received time is later than this timestamp. Note: The time, including time zone, on the entities that generate and validate the signed URI need to be in sync. In the CDNI case, this means that servers at both the CSP, uCDN and dCDN need to be time-synchronized. It is RECOMMENDED to use NTP for this.

- Client IP (CIP) [optional] - IP address, or IP prefix, for which the Signed URI is valid. This is represented in CIDR notation, with dotted decimal format for IPv4 or canonical text representation for IPv6 addresses [RFC5952]. The request is rejected if sourced from a client outside of the specified IP range.

- Original URI Container (OUC) [optional] - Container for holding the Full Original URI while the URI signature is calculated. The Original URI Container information element is not transmitted as part of the URI Signing Package Attribute. If the Original URI Container information element is used, the URI Pattern Sequence information element MUST NOT be used.

- URI Pattern Container (UPC) [optional] - Percent-encoded container for one or more URI Patterns that describes for which content the Signed URI is valid. The URI Pattern Container contains an expression to match against the requested URI to check whether the requested content is allowed to be requested. Multiple URI Patterns may be concatenated in a single URI Pattern Container

information element by separating them with a semi-colon (';') character. Each URI Pattern follows the [RFC3986] URI format, including the '://' that delimits the URI scheme from the hierarchy part. The pattern may include the wildcards '*' and '?', where '*' matches any sequence of characters (including the empty string) and '?' matches exactly one character. The three literals '$', '*' and '?' should be escaped as '$$', '$*' and '$?'. All other characters are treated as literals. The following is an example of a valid URI Pattern: '*://*/folder/content-83112371/quality_*/segment????.mp4'. In its final percent-encoded form, this is equal to '%2A%3A%2F%2F%2A%2Ffolder%2Fcontent-83112371%2Fquality_%2A%2Fsegment%3F%3F%3F%3F.mp4'. An example of two concatenated URI Patterns is the following: 'http://*/folder/content-83112371/manifest/*.xml;http://*/folder/content-83112371/quality_*/segment????.mp4', which in percent-encoded form is: 'http%3A%2F%2F%2A%2Ffolder%2Fcontent-83112371%2Fmanifest%2F%2A.xml%3Bhttp%3A%2F%2F%2A%2Ffolder%2Fcontent-83112371%2Fquality_%2A%2Fsegment%3F%3F%3F%3F.mp4' If the UPC is used, the Original URI Container information element MUST NOT be used.

The Expiry Time Information Element ensures that the content authorization expires after a predetermined time. This limits the time window for content access and prevents replay of the request beyond the authorized time window.

The Client IP Information Element is used to restrict content access to a particular IP address or set of IP addresses based on the IP address for whom the content access was authorized. The URI Signing mechanism described in this document will communicate the IP address in the URI. To prevent the IP address from being logged, the Client IP information element is transmitted in encrypted form.

The Original URI Container is used to limit access to the Original URI only.

The URI Pattern Container Information Element is used to restrict content access to a particular set of URIs.

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

Note: See the Security Considerations (Section 9) section on the limitations of using an expiration time and client IP address for distribution policy enforcement.

2.2. Signature Computation Information Elements

This section identifies the set of information elements that may be needed to verify the URI (signature). New information elements may be introduced in the future if new URI signing algorithms are developed.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to validate the URI by recreating the URI Signature.

- Version (VER) [optional] - An 8-bit unsigned integer used for identifying the version of URI signing method. If this Information Element is not present in the URI Signing Package Attribute, the default version is 1.

- Key ID (KID) [optional] - A string used for obtaining the key (e.g., database lookup, URI reference) which is needed to validate the URI signature. The KID and KID_NUM information elements MUST NOT be present in the same URI Signing Package Attribute.

- Numerical Key ID (KID_NUM) [optional] - A 64-bit unsigned integer used as an optional alternative for KID. The KID and KID_NUM information elements MUST NOT be present in the same URI Signing Package Attribute.

- Hash Function (HF) [optional] - A string used for identifying the hash function to compute the URI signature with HMAC. If this Information Element is not present in the URI Signing Package Attribute, the default hash function is "SHA-256". For interoperability purposes, any hash function signalled via this Information Element SHALL use the notation as used by NIST (e.g. "SHA-256" instead of "SHA256", as defined in [FIPS.180-1.1995]).

- Digital Signature Algorithm (DSA) [optional] - Algorithm used to calculate the Digital Signature. If this Information Element is not present in the URI Signing Package Attribute, the default is "ECDSA". For interoperability purposes, any digital signature algorithm signalled via this Information Element SHALL use the notation as used by NIST (e.g. "ECDSA" instead of "EC-DSA", as defined in [FIPS.186-4.2013]).

- Client IP Encryption Algorithm (CEA) [optional] - Algorithm used to encrypt the Client IP. If this Information Element is not present in the URI Signing Package Attribute, the default is "AES-128". For interoperability purposes, any encryption algorithm signalled via this Information Element SHALL use the notation as used by NIST (e.g. "AES-128" instead of "AES128", as defined in [FIPS.197.2001]").

- Client IP Key ID (CKI) [optional] - A 64-bit unsigned integer used for obtaining the key (e.g., database lookup) used for encrypting/decrypting the Client IP.

The Version Information Element indicates which version of URI signing scheme is used (including which attributes and algorithms are supported). The present document specifies Version 1. If the Version attribute is not present in the Signed URI, then the version is obtained from the CDNI metadata, else it is considered to have been set to the default value of 1. More versions may be defined in the future.

The Key ID Information Element is used to retrieved the key which is needed as input to the algorithm for validating the Signed URI. The method used for obtaining the actual key from the reference included in the Key ID Information Element is outside the scope of this document. Instead of using the KID element, which is a string, it is possible to use the KID_NUM element for numerical Key identifiers instead. The KID_NUM element is a 64-bit unsigned integer. In cases where numerical KEY IDs are used, it is RECOMMENDED to use KID_NUM instead of KID.

The Hash Function Information Element indicates the hash function to be used for HMAC-based message digest computation. The Hash Function Information Element is used in combination with the Message Digest Information Element defined in section Section 2.3.

The Digital Signature Algorithm Information Element indicates the digital signature function to be in the case asymmetric keys are used. The Digital Signature Algorithm Information Element is used in combination with the Digital Signature Information Element defined in section Section 2.3.

The Client IP Encryption Algorithm Information Element indicates the encryption algorithm to be used for the Client IP. The Client IP Encryption Algorithm Information Element is used in combination with the Client IP Information Element defined in section Section 2.1.

The Client IP Key ID is used to retrieved the key which is used for encrypting and decrypting the Client IP. The method used for

obtaining the actual key from the reference included in the Key ID Information Element is outside the scope of this document. The Client IP Encryption Algorithm Information Element is used in combination with the Client IP Information Element defined in section Section 2.1.

2.3. URI Signature Information Elements

This section identifies the set of information elements that carry the URI Signature that is used for checking the integrity and authenticity of the URI.

The defined keyword for each information element is specified in parenthesis below.

The following information elements are used to carry the actual URI Signature.

- Message Digest (MD) [mandatory for symmetric key] - A string used for the message digest generated by the URI signing entity.

- Digital Signature (DS) [mandatory for asymmetric keys] - A string used for the digital signature provided by the URI signing entity.

The Message Digest attribute contains the message digest used to validate the Signed URI when symmetric keys are used.

The Digital Signature attribute contains the digital signature used to verify the Signed URI when asymmetric keys are used.

In the case of symmetric key, HMAC algorithm is used for the following reasons: 1) Ability to use hash functions (i.e., no changes needed) with well understood cryptographic properties that perform well and for which code is freely and widely available, 2) Easy to replace the embedded hash function in case faster or more secure hash functions are found or required, 3) Original performance of the hash function is maintained without incurring a significant degradation, and 4) Simple way to use and handle keys. The default HMAC algorithm used is SHA-256.

In the case of asymmetric keys, Elliptic Curve Digital Signature Algorithm (EC DSA) - a variant of DSA - is used because of the following reasons: 1) Key size is small while still offering good security, 2) Key is easy to store, and 3) Computation is faster than DSA or RSA.

2.4. URI Signing Package Attribute

The URI Signing Package Attribute is an encapsulation container for the URI Signing Information Elements defined in the previous sections. The URI Signing Information Elements are encoded and stored in this attribute. URI Signing Package Attribute is appended to the Original URI to create the Signed URI.

The primary advantage of the URI Signing Package Attribute is that it avoids having to expose the URI Signing Information Elements directly in the query string of the URI, thereby reducing the potential for a namespace collision space within the URI query string (or the URL path in case path parameters are used). A side-benefit of the attribute is the obfuscation performed by the URI Signing Package Attribute hides the information (e.g., client IP address) from view of the common user, who is not aware of the encoding scheme. Obviously, this is not a security method since anyone who knows the encoding scheme is able to obtain the clear text. Note that any parameters appended to the query string after the URI Signing Package Attribute are not validated and hence do not affect URI Signing.

The following attribute is used to carry the encoded set of URI Signing attributes in the Signed URI.

- URI Signing Package (URISigningPackage) - The encoded attribute containing all the CDNI URI Signing Information Elements used for URI Signing.

The URI Signing Package Attribute contains the URI Signing Information Elements in the Base-64 encoding with URL and Filename Safe Alphabet (a.k.a. "base64url") as specified in the Base-64 Data Encoding [RFC4648] document. The URI Signing Package Attribute is the only URI Signing attribute exposed in the Signed URI. If the Signed URI is communicated via the URI query string, the attribute MUST be the last parameter in the query string of the URI when the Signed URI is generated. However, a client or CDN may append other query parameters unrelated to URI Signing to the Signed URI. Such additional query parameters SHOULD NOT use the same name as the URI Signing Package Attribute to avoid namespace collision and potential failure of the URI Signing validation.

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

2.5. User Agent Attributes

For some use cases, such as logging, it might be useful to allow the UA, or another entity, add one or more attributes to the Signed URI for purposes other than URI Signing without causing URI Signing to fail. In order to do so, such attributes MUST be appended after the URI Signing Packacke Attribute. Any attributes appended in such way after the URI Signature has been calculated are not validated for the purpose of content access authorization. Adding any such attributes to the Signed URI before the URI Signing Packacke Attribute will cause the URI Signing validation to fail.

Note that a malicious UA might potentially use the ability to append attributes to the Signed URI in order to try to influence the content that is delivered. For example, the UA might append '&quality=HD' to try to make the dCDN deliver an HD version of the requested content. Since such an additional attribute is appended after the URI Signing Package Attribute it is not validated and will not affect the outcome of the URI validation. In order to deal with this vulnerability, a dCDN is RECOMMENDED to ignore any query strings appended after the URI Signing Package Attribute for the purpose of content selection.

3. Create a Signed URI

The following procedure for signing a URI defines the algorithms in this version of URI Signing. Note that some steps may be skipped if the CSP does not enforce a distribution policy and the Enforcement Information Elements are therefore not necessary. A URI (as defined in URI Generic Syntax [RFC3986]) contains the following parts: scheme name, authority, path, query, and fragment. If the Original URI Container information element is used, all components except for the scheme part are protected by the URI Signature. This allows the URI signature to be validated correctly in the case when a client performs a fallback to another scheme (e.g., HTTP) for a content item referenced by a URI with a specific scheme (e.g., RTSP). In case the URI Pattern Container information element is used, the CSP has full flexibility to specify which elements of the URI (including the scheme part) are protected by the URI.

The process of generating a Signed URI can be divided into four sets of steps: 1) Compose URI Signing IEs with original URI / URI pattern, 2) Compute the URI Signature, 3) Encode the URI Signing Package, and 4) Assemble the parts to create the Signed URI. Note it is possible to use some other algorithm and implementation as long as the same result is achieved. An example for the Full Original URI, "http://example.com/content.mov", is used to clarify the steps.

3.1. Compose URI Signing IEs with Protected URI

Calculate the URI Signature by following the procedure below.

1. Create an empty buffer for performing the operations below.

2. If the version is not the default value (i.e. "1"), perform this step. Specify the version by appending the string "VER=#" to the buffer, where '#' represents the new version number. The following steps in the procedure are based on the initial version of URI Signing specified by this document. For other versions, reference the associated RFC for the URI signing procedure.

3. If time window enforcement is needed, perform this step.

   A. If an information element was added to the buffer, append an "&" character. Append the string "ET=". Note in the case of re-signing a URI, the information element MUST be carried over from the received Signed URI.

   B. Get the current time in seconds since epoch (as an integer). Add the validity time in seconds as an integer. Note in the case of re-signing a URI, the value MUST remain the same as the received Signed URI.

   C. Convert this integer to a string and append to the buffer.

4. If client IP enforcement is needed, perform this step.

   A. Skip this step when the Client IP Encryption Algorithm used is the default ("AES-128"). If an information element was added to the buffer, append an "&" character. Append the string "CEA=". Append the string for the Client IP Encryption Algorithm to be used.

   B. If the Client IP Key Identifier is needed, perform this step. If an information element was added to the buffer, append an "&" character. Append the string "CKI=". Append the Client IP key identifier (e.g., "56128239") needed by the entity to locate the shared key for decrypting the Client IP.

   C. If an information element was added to the buffer, append an "&" character. Append the string "CIP=".

   D. Convert the client's IP address in CIDR notation (dotted decimal format for IPv4 or canonical text representation for IPv6 [RFC5952]) to a string and encrypt it using AES-128 (in ECB mode) or another algorithm if specified by the CEA

Information Element. Note in the case of re-signing a URI, the client IP that is encrypted MUST be equal to the unencrypted value of the Client IP as received in the Signed URI, see step 1 in Section 4.5.

E. Convert the encrypted Client IP to its equivalent hexadecimal format.

F. Append the value computed in the previous step to the buffer.

5. If a Key ID information element is needed, perform this step. If an information element was added to the buffer, append an "&" character. Append the string "KID=" in case a string-based Key ID is used, or "KID_NUM=" in case a numerical Key ID is used. Append the key identifier (e.g. "example:keys:123" or "56128239") needed by the entity to locate the shared key for validating the URI signature.

6. If symmetric shared key is used, perform this step. However, skip this step when the hash function for the HMAC uses the default value ("SHA-256"). If an information element was added to the buffer, append an "&" character. Append the string "HF=". Append the string for the new hash function to be used. Note that re-signing a URI MUST use the same hash function as the received Signed URI or one of the allowable hash functions designated by the CDNI metadata.

7. If asymmetric private/public keys are used, perform this step. However, skip this step when the digital signature algorithm uses the default value ("ECDSA"). If an information element was added to the buffer, append an "&" character. Append the string "DSA=". Append the string for the digital signature function. Note that re-signing a URI MUST use the same digital signature algorithm as the received Signed URI or one of the allowable digital signature algorithms designated by the CDNI metadata.

8. Depending on the type of URI enforcement used (Full Original URI or URI Pattern), add the appropriate information element.

A. If enforcement based on the Full Original URI, perform this step. If an information element was added to the buffer, append an "&" character. Append the string "OUC=". Append the Original URI, excluding the "scheme name" part and the "://" delimiter, to the buffer. Note: the Original URI Container information element MUST be the last information element in the buffer before the signature information element.

B. If enforcement based on a URI Pattern, perform this step. If an information element was added to the buffer, append an "&" character. Append the string "UPC=". Append the URI Pattern Container in the form of a percent-encoded string to the buffer.

3.2. Compute URI Signature

Compute the URI Signature by following the procedure below. The buffer from the previous section is used.

1. If symmetric shared key is used, perform this step.

   A. Obtain the shared key to be used for signing the URI.

   B. Append the string "MD=". The buffer now contains the complete section of the URI that is protected (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&OUC=example.com/content.mov&MD=").

   C. Compute the message digest using the HMAC algorithm and the default SHA-256 hash function, or another hash function if specified by the HF Information Element, with the shared key and message as the two inputs to the hash function.

   D. Convert the message digest to its equivalent hexadecimal format.

   E. Append the string for the message digest (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&OUC=example.com/content.mov&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").

2. If asymmetric private/public keys are used, perform this step.

   A. Obtain the private key to be used for signing the URI.

   B. If an information element was added to the buffer, append an "&" character. Append the string "DS=". The buffer now contains the complete section of the URI that is protected. (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&OUC=example.com/content.mov&DS=").

   C. Compute the message digest using SHA-1 (without a key) for the buffer. Note: The digital signature generated in the next step is calculated over the SHA-1 message digest, instead of over the full cleartext buffer. This is done to reduce the length of the digital signature, the URI Signing

Package Attribute, and the resulting Signed URI. Since SHA-1 is not used for cryptographic purposes here, the security concerns around SHA-1 do not apply.

D. Compute the digital signature, using the EC-DSA algorithm by default, or another algorithm if specified by the DSA Information Element, with the private EC key and message digest (obtained in previous step) as inputs.

E. Convert the digital signature to its equivalent hexadecimal format.

F. Append the string for the digital signature. In the case where EC-DSA algorithm is used, this string contains the values for the 'r' and 's' parameters, delimited by ':' (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&OUC=example.com/content.mov&DS=r:CFB03EDB33810AB6C79EE3C47FBD86D227D702F25F66C01CF03F59F1E005668D:s:57ED0E8DF7E786C87E39177DD3398A7FB010E6A4C0DC8AA71331A929A29EA24E")

3.3. Encode the URI Signing Package

Encode the URI Signing Package by following the procedure below. The buffer from the previous section is used.

1. If enforcement is based on the Full Original URI, this step is performed. Remove the Original URI Container Attribute from the buffer, including the preceding "&" character (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf"). Note: This attribute is not needed in the encoded URI Signing Package because the Full Original URI is part of the Signed URI.

2. Compute the URI Signing Package Attribute using Base-64 Data Encoding [RFC4648] on the message (e.g. "RVQ9MTIwOTQyMjk3NiZhbXA7Q0tJPTMxMSZhbXA7Q0lQPTkwQzkxMzk3NzkzM0ZDNjUwRTcxODYzNjFBOTNENkMzJmFtcDtLSUQ9ZXhhbXBsZTprZXlzOjEyMyZhbXA7TUQ9MWVjYjE0NDZhNjQzMTM1MmFhYjBmYjZlMGRjYTMwZTMwMzU2NTkzYTk3YWNiOTcyMjAyMTIwZGM0ODJiZGRhZg=="). Note: This is the value for the URI Signing Package Attribute.

3.4. Assemble the Signed URI

Assemble the parts to create the Signed URI by following the procedure below.

1. Copy the entire Full Original URI into a new empty buffer.

2. If the Signed URI is communicated via the URI query string, perform this step.

   A. Check if the Full Original URI already contains a query string. If not, append a "?" character. If yes, append an "&" character.

   B. Append the parameter name used to indicate the URI Signing Package Attribute, as communicated via the CDNI Metadata interface, followed by an "=". If none is communicated by the CDNI Metadata interface, it defaults to "URISigningPackage". For example, if the CDNI Metadata interface specifies "SIG", append the string "SIG=" to the message.

   C. Append the URI Signing Package that was generated in previous section (e.g. "http://example.com/content.mov?URISigningPackage=RVQ9MTIwOTQyMjk3NiZhbXA7Q0tJPTMxMSZhbXA7Q0lQPTkwQzkxMzk3NzkzM0ZDNjUwRTcxODYzNjFBOTNENkMzJmFtcDtLSUQ9ZXhhbXBsZTprZXlzOjEyMyZhbXA7TUQ9MWVjYjE0NDZhNjQzMTM1MmFhYjBmYjZlMGRjYTMwZTMwMzU2NTkzYTk3YWNiOTcyMjAyMTIwZGM0ODJiZGRhZg=="). Note: this is the completed Signed URI.

3. If the Signed URI is communicated via a URL path parameter, perform this step.

   A. Check if the Full Original URI already contains a path parameter. If not, add "/;" before the last path component indicating the file to be retrieved. If yes, character at the last append a "?" character. If yes, append an ";" character after the last path parameter.

   B. Append the parameter name used to indicate the URI Signing Package Attribute, as communicated via the CDNI Metadata interface, after the inserted ";" character. If none is communicated by the CDNI Metadata interface, it defaults to "URISigningPackage". Append an "=" character. For example, if the CDNI Metadata interface specifies "SIG" as the parameter name, append the string "SIG=" to the message.

   C. Append the URI Signing Package that was generated in previous section after the "=" character (e.g. "http://example.com/;URISigningPackage=RVQ9MTIwOTQyMjk3NiZhbXA7Q0tJPTMxMSZhbXA7Q0lQPTkwQzkxMzk3NzkzM0ZDNjUwRTcxODYzNjFBOTNENkMzJmFtcDtLSUQ9ZXhhbXBsZTprZXlzOjEyMyZhbXA7TUQ9MWVjYjE0NDZhNjQzMTM1MmFhYjBmYjZlMGR

jYTMwZTMwMzU2NTkzYTk3YWNiOTcyMjAyMTIwZGM0ODJiZGRhZg==/content.mov"). Note: this is the completed Signed URI.

4. Validate a Signed URI

The process of validating a Signed URI can be divided into five sets of steps: 1) Extract and decode URI Signing Package from the Signed URI, 2) Extract the URI Signing information elements, 3) Obtain the Protected URI, 4) Validate URI signature to ensure integrity of Signed URI, and 5) Ensure proper enforcement of the distribution policy. The integrity of the Signed URI is confirmed before distribution policy enforcement because validation procedure will detect first if the URI has been tampered with. Note it is possible to use some other algorithm and implementation as long as the same result is achieved.

4.1. Extract and Decode URI Signing Package

Extract the encoded URI Signing Package Attribute from the Signed URI. The attribute is decoded for subsequent processing by the Downstream CDN.

1. Extract the value from 'URISigningPackage' attribute. This value is the encoded URI Signing Package Attribute. If there are multiple instances of this attribute, the first one is used and the remaining ones are ignored. This ensures that the Signed URI can be validated despite a client appending another instance of the 'URISigningPackage' attribute.

2. Decode the string using Base-64 Data Encoding [RFC4648] to obtain all the URI Signing information elements (e.g. "ET=1209422976&CKI=311&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&MD=1ecb1446a6431352aab0fb6e0dca30e30356593a97acb972202120dc482bddaf").

4.2. Extract URI Signing IEs

Extract the information elements in the URI Signing Package Attribute. Note that some steps are to be skipped if the corresponding URI Signing information elements are not embedded in the attribute. Some of the information elements will be used to validate the URI signature in the subsequent section.

1. Extract the value from "VER" if the information element exists in the decoded URI Signing Package. Determine the version of the URI Signing algorithm used to process the Signed URI. If the CDNI Metadata interface is used, check to see if the used version of the URI Signing algorithm is among the allowed set of

URI Signing versions specified by the metadata. If this is not the case, the request is denied. If the information element is not in the URI, then obtain the version number in another manner (e.g., configuration, CDNI metadata or default value).

2. Extract the value from "MD" if the information element exists in the decoded URI Signing Package. The existence of this information element indicates a symmetric key is used.

3. Extract the value from "DS" if the information element exists in the decoded URI Signing Package. The existence of this information element indicates an asymmetric key is used.

4. If neither "MD" or "DS" attribute is in the decoded URI Signing Package, then no URI Signature exists and the request is denied. If both the "MD" and the "DS" information elements are present, the Signed URI is considered to be malformed and the request is denied.

5. Extract the value from "UPC" if the information element exists in the decoded URI Signing Package and convert it from its percent-encoded form to a regular string. The existence of this information element indicates content delivery is enforced based on a (set of) URI pattern(s) instead of the Full Original URI.

6. Extract the value from "CIP" if the information element exists in the decoded URI Signing Package. The existence of this information element indicates content delivery is enforced based on client IP address.

7. Extract the value from "ET" if the information element exists in the decoded URI Signing Package. The existence of this information element indicates content delivery is enforced based on time.

8. Extract the value from the "KID" or "KID_NUM" information element, if they exist. The existence of either of these information elements indicates a key can be referenced. If both the "KID" and the "KID_NUM" information elements are present, the Signed URI is considered to be malformed and the request is denied.

9. Extract the value from the "HF" information element, if it exists. The existence of this information element indicates a different hash function than the default.

10. Extract the value from the "DSA" information element, if it exists. The existence of this information element indicates a different digital signature algorithm than the default.

11. Extract the value from the "CEA" information element, if it exists. The existence of this information element indicates a different Client IP Encryption Algorithm than the default.

12. Extract the value from the "CKI" information element, if it exists. The existence of this information element indicates a key can be referenced using which the Client IP was encrypted.

4.3. Obtain URI Signing IEs with Protected URI

Obtain the message that contains the URI Signing Information Elements and Protected URI (either Full Original URI or URI pattern). This is the content that was used to generate the URI signature, which is validated by Downstream CDN in the next section.

1. Copy the decoded URI Signing Package into a new buffer to hold the message for performing the operations below. Note: The attribute contains all the URI Signing Information Elements and may also include the URI Pattern Container.

2. Remove the value part of the "MD" or "DS" information element from the message. The part of information element that remains is "MD=" or "DS=".

3. When UPC information element exists, the Protected URI is a set of URIs (i.e., URI Pattern which is conveyed in the value of the UPC IE). Otherwise, the Protected URI is the Full Original URI.

    A. For URI Pattern, the message already contains the Protected URI. Therefore, no additional operation is needed to create the protected URI.

    B. For Full Original URI, the message is missing the Full Original URI in the URI Signing Package. Perform the following steps.

        1. Remove the string "MD=" or "DS=".

        2. Append the string "OUC=". Append the Full Original URI, excluding the "scheme name" part and the "://" delimiter, to the buffer.

        3. Append the "&" character. Append "MD=" or "DS=", depending on which of the two was present in the URI

Signing Package. The message is ready for validation of the message digest (e.g. "ET=1209422976&CIP=90C913977933FC650E7186361A93D6C3&KID=example:keys:123&OUC=example.com/content.mov&MD=").

4.4. Validate URI Signature

Validate the URI Signature for the Signed URI. The message used for computation is obtained from previous section.

1. The received message signature is the value extracted from the "MD" or "DS" information element. Convert the message signature to binary format. This will be used to compare with the computed value later.

2. Based on the presence of either the MD or DS information element in the URI Signing Package, validate the message digest or digital signature for symmetric key or asymmetric keys, respectively.

   A. For MD, an HMAC algorithm is used.

      1. If either the "KID" or "KID_NUM" information element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or "KID_NUM" information element is present in the Signed URI, obtain the shared key via CDNI metadata or configuration.

      2. If "HF" information element exists, validate that the hash function is in the allowable "HF" set as listed in the CDNI metadata or configuration. The request is denied when the hash function is not allowed. Otherwise, the "HF" information element is not in the Signed URI. In this case, the default hash function is SHA-256.

      3. Compute the message digest using the HMAC algorithm with the shared key and message as the two inputs to the hash function.

      4. Compare the result with the received message signature to validate the Signed URI.

   B. For DS, a digital signature function is used.

1. If either the "KID" or "KID_NUM" information element exists, validate that the key identifier is in the allowable KID set as listed in the CDNI metadata or configuration. The request is denied when the key identifier is not allowed. If neither the "KID" or "KID_NUM" information element is present in the Signed URI, obtain the public key via CDNI metadata or configuration.

2. If "DSA" information element exists, validate that the digital signature algorithm is in the allowable "DSA" set as listed in the CDNI metadata or configuration. The request is denied when the DSA is not allowed. Otherwise, the "DSA" information element is not in the Signed URI. In this case, the default DSA is EC-DSA.

3. Compute the message digest using SHA-1 (without a key) for the message.

4. Verify the digital signature using the digital signature function (e.g., EC-DSA) with the public key, received digital signature, and message digest (obtained in previous step) as inputs. This validates the Signed URI.

4.5. Distribution Policy Enforcement

Note that the absence of a given Enforcement Information Element indicates enforcement of its purpose is not necessary in the CSP's distribution policy.

1. If the "CIP" information element does not exist, this step can be skipped.

   A. Obtain the key for decrypting the Client IP, as indicated by the Client IP Key Index information element or set via configuration.

   B. Decrypt the encrypted Client IP address obtained in step 6 using AES-128, or the algorithm specified by the Client IP Encryption Algorithm information element.

   C. Verify, using CIDR matching, that the request came from an IP address within the range indicated by the decrypted Client IP information element. If the IP address is incorrect, the request is denied.

2. If the "ET" information element exists, validate that the request arrived before expiration time based on the "ET" information element. If the time expired, then the request is denied.

3. If the "UPC" information element exists, validate that the requested resource is in the allowed set by matching the received URI against each of the Patterns in the URI Pattern Container information element until a match is found. If there is no match, the request is denied.

5. Relationship with CDNI Interfaces

Some of the CDNI Interfaces need enhancements to support URI Signing. As an example: A Downstream CDN that supports URI Signing needs to be able to advertise this capability to the Upstream CDN. The Upstream CDN needs to select a Downstream CDN based on such capability when the CSP requires access control to enforce its distribution policy via URI Signing. Also, the Upstream CDN needs to be able to distribute via the CDNI Metadata interface the information necessary to allow the Downstream CDN 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?).

5.1. CDNI Control Interface

URI Signing has no impact on this interface.

5.2. CDNI Footprint & Capabilities Advertisement Interface

The Downstream CDN advertises its capability to support URI Signing via the CDNI Footprint & Capabilities Advertisement interface (FCI). The supported version of URI Signing needs to be included to allow for future extensibility.

In general, new information elements introduced to enhance URI Signing requires a draft and a new version.

For Enforcement Information Elements, there is no need to advertise the based information elements such as "CIP" and "ET".

For Signature Computation Information Elements:

No need to advertise "VER" Information Element unless it's not "1". In this case, a draft is needed to describe the new version.

Advertise value of the "HF" Information Element (i.e. SHA-256) to indicate support for the hash function; Need IANA assignment for new hash function.

Advertise value of the "DSA" Information Element (i.e. "ECDSA") to indicate support for the DSA; Need IANA assignment for new digital signature algorithm.

Advertise "MD" Information Element (i.e., SHA-256) to indicate support for symmetric key method; A new draft is needed for an alternative method.

Advertise "DS" Information Element (i.e., "ECDSA") to indicate support for asymmetric key method; A new draft is needed for an alternative method.

For URI Signing Package Attribute, there is no need to advertise the base attribute.

5.3. CDNI Request Routing Redirection Interface

The CDNI Request Routing Redirection Interface [I-D.ietf-cdni-redirection] describes the recursive request redirection method. For URI Signing, the Upstream CDN signs the URI provided by the Downstream CDN. This approach has the following benefits:

Consistency with interative request routing method

URI Signing is fully operational even when Downstream CDN does not have the signing function (which may be the case when the Downstream CDN operates only as a delivering CDN)

Upstream CDN can act as a conversion gateway for the requesting routing interface between Upstream CDN and CSP and request routing interface between Upstream CDN and Downstream CDN since these two interfaces may not be the same

5.4. CDNI Metadata Interface

The CDNI Metadata Interface [I-D.ietf-cdni-metadata] describes the CDNI metadata distribution in order to enable content acquisition and delivery. For URI Signing, additional CDNI metadata objects are specified. In general, an Empty set means "all". These are the CDNI metadata objects used for URI Signing.

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 Downstream CDN to ensure that the URI must be signed and validated before content delivery. Otherwise, Downstream CDN does not perform validation regardless if URI is signed or not.

Type: Boolean

Mandatory-to-Specify: No. If a UriSigning object is present in the metadata for a piece of content (even if the object is empty), then URI signing should be enforced. If no UriSigning object is present in the metadata for a piece of content, then the URI signature should not be validated.

Property: key-id

Description: Designated key identifier used for URI Signing computation when the Signed URI does not contain the Key ID information element.

Type: String

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

Property: key-id-set

Description: Allowable Key ID set that the Signed URI’s Key ID information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any Key ID.

Property: hash-function

Description: Designated hash function used for URI Signing computation when the Signed URI does not contain the Hash Function information element.

Type: String (limited to the hash function strings in the registry defined by the IANA Considerations (Section 8) section)

Mandatory-to-Specify: No. Default is SHA-256.

Property: hash-function-set

Description: Allowable Hash Function set that the Signed URI's Hash Function information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any hash function.

Property: digital-signature-algorithm

Description: Designated digital signature function used for URI Signing computation when the Signed URI does not contain the Digital Signature Algorithm information element.

Type: String (limited to the digital signature algorithm strings in the registry defined by the IANA Considerations (Section 8) section).

Mandatory-to-Specify: No. Default is "ECDSA".

Property: digital-signature-algorithm-set

Description: Allowable digital signature function set that the Signed URI's Digital Signature Algorithm information element can reference.

Type: List of Strings

Mandatory-to-Specify: No. Default is to allow any DSA.

Property: version

Description: Designated version used for URI Signing computation when the Signed URI does not contain the VER attribute.

Type: Integer

Mandatory-to-Specify: No. Default is 1.

Property: version-set

Description: Allowable version set that the Signed URI's VER attribute can reference.

Type: List of Integers

Mandatory-to-Specify: No. Default is to allow any version.

Property: package-attribute

Description: Overwrite the default name for the URL Signing Package Attribute.

Type: String

Mandatory-to-Specify: No. Default is "URISigningPackage".

Note that the Key ID information element is not needed if only one key is provided by the CSP or the Upstream CDN for the content item or set of content items covered by the CDNI Metadata object. In the case of asymmetric keys, it's easy for any entity to sign the URI for content with a private key and provide the public key in the Signed URI. This just confirms that the URI Signer authorized the delivery. But it's necessary for the URI Signer to be the content owner. So, the CDNI Metadata interface or configuration MUST provide the allowable Key ID set to authorize the Key ID information element embedded in the Signed URI.

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

```
{
  "generic-metadata-type": "MI.UriSigning.v1"
  "generic-metadata-value": {}
}
```

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

```
{
  "generic-metadata-type": "MI.UriSigning.v1"
  "generic-metadata-value":
    {
      "enforce": true,
      "key-id": "1",
      "key-id-set": ["1", "2", "3"],
      "hash-function": "SHA-512",
      "hash-function-set": ["SHA-384", "SHA-512"],
      "digital-signature-algorithm": "ECDSA",
      "digital-signature-algorithm-set": ["ECDSA"],
      "version": 1,
      "version-set": [1],
      "package-attribute": "usp"
    }
}
```

5.5. CDNI Logging Interface

For URI Signing, the Downstream CDN 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 [I-D.ietf-cdni-logging].

- 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 URI signature validation performed
    - "200" : URI signature validation performed and validated
    - "400" : URI signature validation performed and rejected because of incorrect signature
    - "401" : URI signature validation performed and rejected because of Expiration Time enforcement

+ "402" : URI signature validation performed and rejected because of Client IP enforcement

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

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

+ "501" : unable to perform URI signature validation because of unsupported version number

* 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 URI signature was rejected, e.g., for debugging purposes.

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

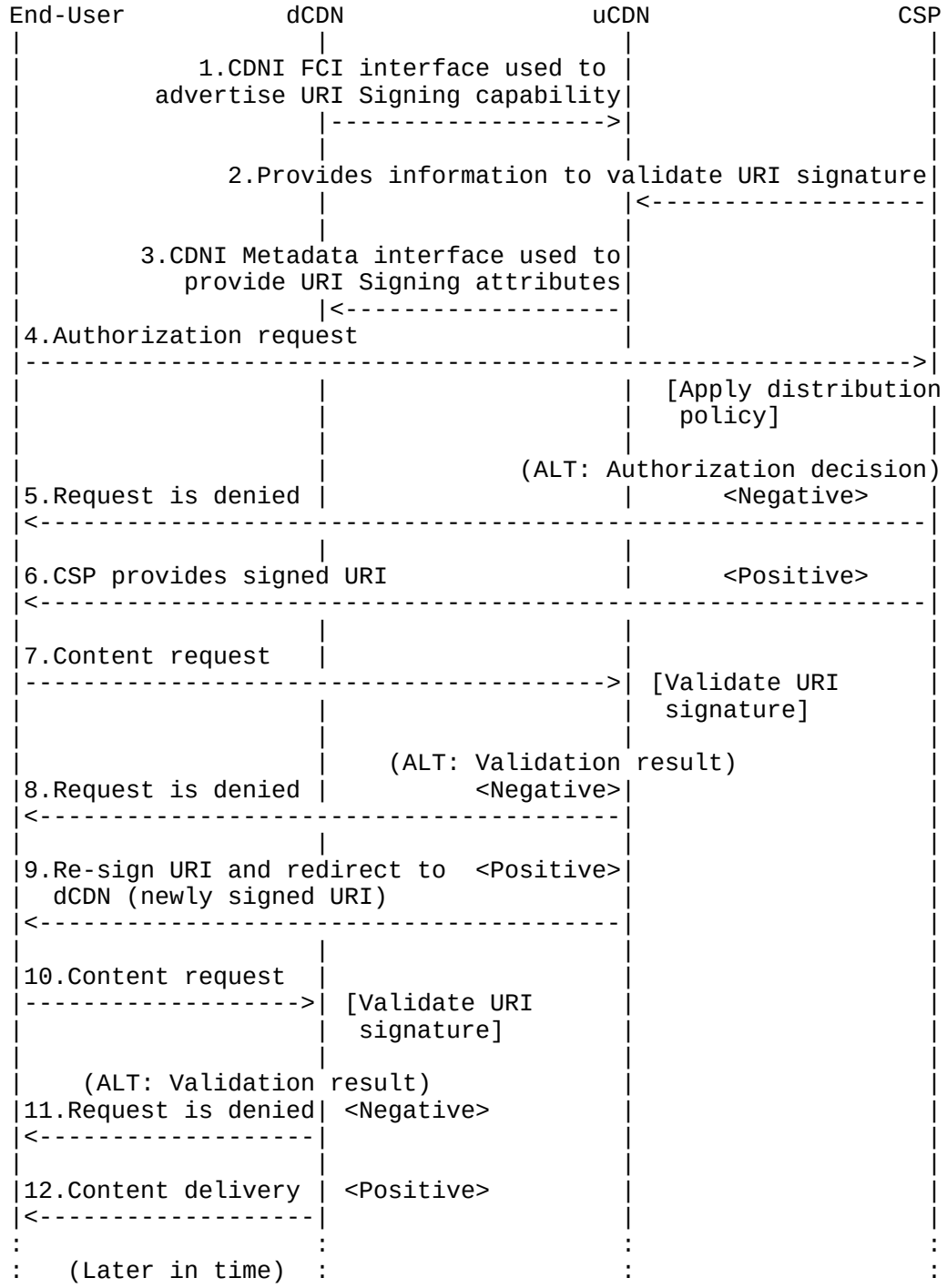
6. URI Signing Message Flow

URI Signing supports both HTTP-based and DNS-based request routing. HMAC [RFC2104] defines a hash-based message authentication code allowing two parties that share a symmetric key or asymmetric keys to establish the integrity and authenticity of a set of information (e.g., a message) through a cryptographic hash function.

6.1. HTTP Redirection

For HTTP-based request routing, HMAC is applied to a set of information that is unique to a given end user content request using key information that is specific to a pair of adjacent CDNI hops (e.g. between the CSP and the Authoritative CDN, between the Authoritative CDN and a Downstream CDN). This allows a CDNI hop to ascertain the authenticity of a given request received from a previous CDNI hop.

The URI signing scheme described below is based on the following steps (assuming HTTP redirection, iterative request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.


End-User
dCDN
uCDN
CSP
1.CDNI FCI interface used to advertise URI Signing capability
2.Provides information to validate URI signature
3.CDNI Metadata interface used to provide URI Signing attributes
4.Authorization request
[Apply distribution policy]
(ALT: Authorization decision)
5.Request is denied
<Negative>
6.CSP provides signed URI
<Positive>
7.Content request
[Validate URI signature]
(ALT: Validation result)
8.Request is denied
<Negative>
9.Re-sign URI and redirect to dCDN (newly signed URI)
<Positive>
10.Content request
[Validate URI signature]
(ALT: Validation result)
11.Request is denied
<Negative>
12.Content delivery
<Positive>
(Later in time)

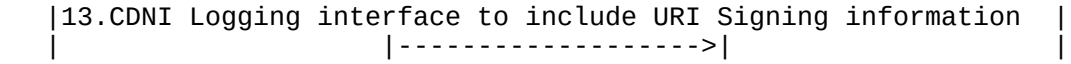


Figure 3: HTTP-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.

2. CSP provides to the Authoritative CDN the information needed to validate URI signatures from that CSP. For example, this information may include a hashing function, algorithm, and a key value.

3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN the information needed to validate URI signatures from the Authoritative CDN for the given CSP. For example, this information may include the URI query string parameter name for the URI Signing Package Attribute, a hashing algorithm and/or a key corresponding to the trust relationship between the Authoritative CDN and the Downstream CDN.

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 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 authoritative CDN validates the URI Signature in the URI using the information provided by the CSP.

8. If the validation is negative, the authoritative CDN rejects the request

9. If the validation is positive, the authoritative CDN computes a Signed URI that is based on unique parameters of that request and provides to the end user as the URI to use to further request the content from the Downstream CDN

10. On receipt of the corresponding content request, the Downstream CDN validates the URI Signature in the Signed URI using the information provided by the Authoritative CDN in the CDNI Metadata

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

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

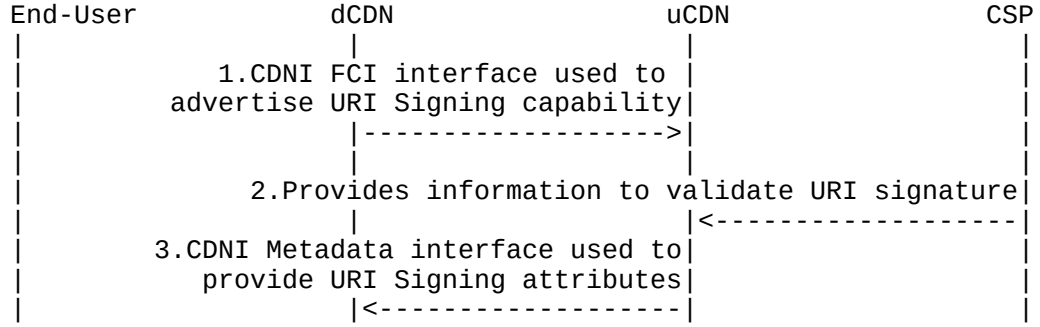
13. At a later time, Downstream CDN reports logging events that includes URI signing information.

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

6.2. DNS Redirection

For DNS-based request routing, the CSP and Authoritative CDN 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 Downstream CDNs. 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 Downstream CDNs. Authorized Delivery CDNs need to obtain the key information to validate the Signed UR, which is computed by the CSP based on its distribution policy.

The URI signing scheme described below is based on the following steps (assuming iterative DNS request routing and a CDN path with two CDNs). Note that Authoritative CDN and Upstream CDN are used exchangeably.



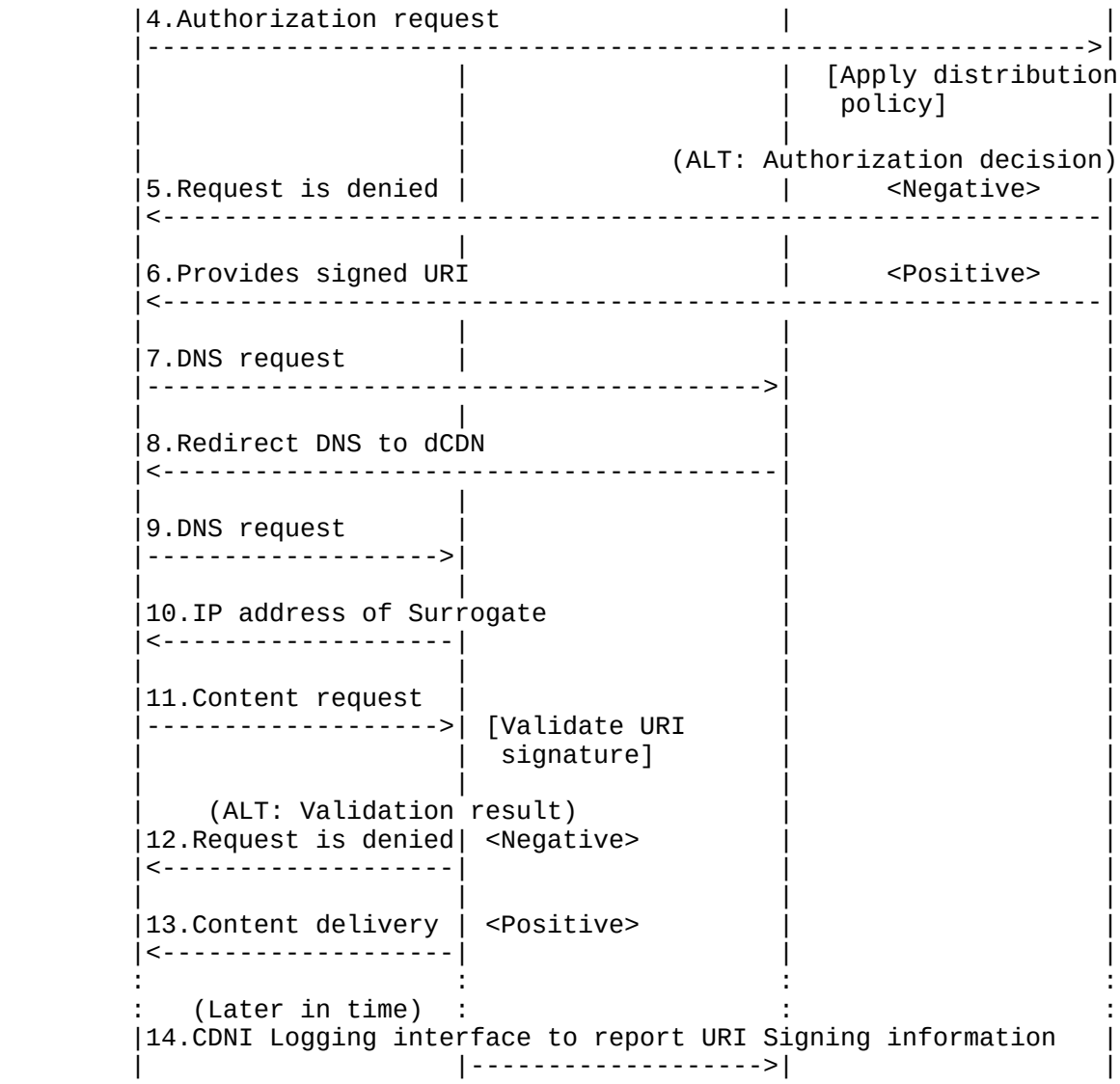


Figure 4: DNS-based Request Routing with URI Signing

1. Using the CDNI Footprint & Capabilities Advertisement interface, the Downstream CDN advertises its capabilities including URI Signing support to the Authoritative CDN.

2. CSP provides to the Authoritative CDN the information needed to validate cryptographic signatures from that CSP. For example,

this information may include a hash function, algorithm, and a key.

3. Using the CDNI Metadata interface, the Authoritative CDN communicates to a Downstream CDN 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 Authoritative CDN checks if the Downstream CDN 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 authoritative CDN.

8. On receipt of the DNS request, the authoritative CDN redirects the request to the Downstream CDN.

9. End user sends DNS request to the Downstream CDN.

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

11. On receipt of the corresponding content request, the Downstream CDN validates the cryptographic signature in the URI using the information provided by the Authoritative CDN in the CDNI Metadata

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

13. If the validation is positive, the Downstream CDN serves the request and delivers the content.

14. At a later time, Downstream CDN 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 need to be distributed across multiple CDNI hops including non-adjacent hops is the public key, that 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 including non-adjacent hops. This raises a security concern for applicability of URI Signing with symmetric keys in case of DNS-based inter-CDN request routing.

7. HTTP Adaptive Streaming

The authors note that in order to perform URI signing for individual content segments of HTTP Adaptive Bitrate content, specific URI signing mechanisms are needed. Such mechanisms are currently out-of-scope of this document. More details on this topic is covered in Models for HTTP-Adaptive-Streaming-Aware CDNI [RFC6983]. In addition, [I-D.brandenburg-cdni-uri-signing-for-has] provides an extension to the algorithm defined in this document that deals specifically with URI signing of segmented content.

8. IANA Considerations

8.1. 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.v1	RFCthis

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

8.1.1. 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 5.4

8.2. 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.]

8.2.1. 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 [I-D.ietf-cdni-logging] plus the two additional fields "s-uri-signing" and "s-uri-signing-deny-reason", registered by this document in Section 8.3. The name, format, field value, and occurence information for the two new fields can be found in Section 5.5 of this document.

8.3. 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.]

8.4. CDNI Metadata Auth Type

This document requests the registration of the following CDNI Metadata Auth type under the IANA "CDNI Metadata Auth Types" registry:

Auth type	Description	Specification
MI.UriSigning.v1	URI Signing version 1	RFCthis

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

8.5. CDNI URI Signing Enforcement Information Elements

The IANA is requested to create a new "CDNI URI Signing Enforcement Information Elements" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Enforcement Information Elements" namespace defines the valid Enforcement Information Elements that may be included in a URI Signing token. Additions to the Enforcement Information Elements namespace conform to the "Specification Required" policy as defined in [RFC5226].

The following table defines the initial Enforcement Information Elements:

Element	Description	RFC
ET	Expiry Time	RFCthis
CIP	Client IP Address	RFCthis
OUC	Original URI Container	RFCthis
URI Pattern Container	Client IP Address	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?]

8.6. CDNI URI Signing Signature Computation Information Elements

The IANA is requested to create a new "CDNI URI Signing Signature Computation Information Elements" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Signature Computation Information Elements" namespace defines the valid Signature Computation Information Elements that may be included in a URI Signing token. Additions to the Signature Computation Information Elements namespace conform to the "Specification Required" policy as defined in [RFC5226].

The following table defines the initial Signature Computation Information Elements:

Element	Description	RFC
VER	Version Number	RFCthis
KID	Non-numerical Key Identifier	RFCthis
KID_NUM	Numerical Key Identifier	RFCthis
HF	Hash Function	RFCthis
DSA	Digital Signature Algorithm	RFCthis
CEA	Client IP Encryption Algorithm	RFCthis
CKI	Client IP Encryption Key Identifier	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?]

8.7. CDNI URI Signing Signature Information Elements

The IANA is requested to create a new "CDNI URI Signing Signature Information Elements" subregistry in the "Content Delivery Networks Interconnection (CDNI) Parameters" registry. The "CDNI URI Signing Signature Information Elements" namespace defines the valid Signature Information Elements that may be included in a URI Signing token. Additions to the Signature Information Elements namespace conform to the "Specification Required" policy as defined in [RFC5226].

The following table defines the initial Signature Information Elements:

Element	Description	RFC
MD	Message Digest for Symmetric Key	RFCthis
DS	Digital Signature for Asymmetric Keys	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?]

9. Security Considerations

This document describes the concept of URI Signing and how it can be used to provide access authorization in the case of interconnected CDNs (CDNI). The primary goal of URI Signing is to make sure that only authorized UAs are able to access the content, with a Content Service Provider (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 Enforcement Information Elements in the URI. The current version of this document includes elements for enforcing Client IP Address 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: Any (valid) Signed URI can be used to perform replay attacks. The vulnerability to replay attacks can be reduced by picking a relatively short window for the Expiration Time attribute, although this is limited by the fact that any HTTP-based request needs a window of at least a couple of seconds to prevent any sudden network issues from preventing legitimate UAs access to the content. One way to reduce exposure to replay attacks is to include in the URI a unique one-time access ID. Whenever the Downstream CDN receives a request with a given unique access ID, it adds that access ID to the list of ’used’ IDs. In the case an illegitimate UA tries to use the same URI through a replay attack, the Downstream CDN can deny the request based on the already-used access ID.

Illegitimate client 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 Downstream CDN. This results in the Downstream CDN 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 that can be found in the HTTP headers.

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

In the case where asymmetric keys are used, the KID information element might contain the URL to the public key. To prevent malicious clients from signing their own URIs and inserting the associated public key URL in the KID field, thereby passing URI validation, it is important that CDNs check whether the URI conveyed in the KID field is in the allowable set of KIDs as listed in the CDNI metadata or set via configuration.

10. Privacy

The privacy protection concerns described in CDNI Logging Interface [I-D.ietf-cdni-logging] apply when the client's IP address (CIP attribute) is embedded in the Signed URI. For this reason, the mechanism described in Section 3.1 encrypts the Client IP before including it in the URI Signing Package (and thus the URL itself).

11. 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, Phil Sorber and Gancho Tenev. In addition, Matt Caulfield provided content for the CDNI Metadata Interface section.

12. References

12.1. Normative References

[I-D.ietf-cdni-logging] Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-27 (work in progress), June 2016.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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, <http://www.rfc-editor.org/info/rfc6707>.

12.2. Informative References

[FIPS.180-1.1995]
National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1, April 1995, <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

[FIPS.186-4.2013]
National Institute of Standards and Technology, "Digital Signature Standard", FIPS PUB 186-1, December 1998, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.184-4.pdf>.

[FIPS.197.2001]
National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[I-D.brandenburg-cdni-uri-signing-for-has]
Brandenburg, R., "URI Signing for HTTP Adaptive Streaming (HAS)", draft-brandenburg-cdni-uri-signing-for-has-03 (work in progress), June 2016.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-18 (work in progress), June 2016.

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-18 (work in progress), April 2016.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[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, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

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

[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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7337>.

Authors' Addresses

Kent Leung
Cisco Systems
3625 Cisco Way
San Jose 95134
USA

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

Francois Le Faucheur
Cisco Systems
Greenside, 400 Avenue de Roumanille
Sophia Antipolis 06410
France

Phone: +33 4 97 23 26 19
Email: flefauch@cisco.com

Ray van Brandenburg
TNO
Anna van Buerenplein 1
Den Haag 2595DC
the Netherlands

Phone: +31 88 866 7000
Email: ray.vanbrandenburg@tno.nl

Bill Downey
Verizon Labs
60 Sylvan Road
Waltham, Massachusetts 02451
USA

Phone: +1 781 466 2475
Email: william.s.downey@verizon.com

Michel Fisher
Limelight Networks
222 S Mill Ave
Tempe, AZ 85281
USA

Phone: +1 360 419 5185
Email: mfisher@llnw.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 24, 2016

K. Ma
Ericsson
J. Seedorf
NEC

April 22, 2016

CDNI Footprint & Capabilities Advertisement Interface
draft-ma-cdni-capabilities-09

Abstract

Content Distribution Network Interconnection (CDNI) is predicated on the ability of downstream CDNs (dCDNs) to handle end-user requests in a functionally equivalent manner to the upstream CDN (uCDN). The uCDN must be able to assess the ability of the dCDN to handle individual requests. The CDNI Footprint & Capabilities Advertisement interface (FCI) is provided for the advertisement of capabilities and the footprints to which they apply by the dCDN to the uCDN. This document describes an approach to implementing the CDNI FCI.

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 http://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 October 24, 2016.

Table of Contents

1. Introduction

The need for footprint and capabilities advertisement in Content Distribution Network Interconnection (CDNI) is described in the CDNI requirements document [RFC7337]. Requirements FCI-1 and FCI-2 describe the need to allow dCDNs to communicate capabilities to the uCDN. Requirement FCI-3 describes how a uCDN may aggregate the footprint and capabilities information for all cascaded dCDNs and use the aggregated information in advertisements to CDNs further upstream. This concept of aggregation can apply to both organizationally different dCDNs (e.g., other CDN providers, or different business units within a larger organization) or logical entities within the same CDN (e.g., using multiple request routers for scalability reasons, to segregate surrogates based on specific protocol support, or to segregate surrogates based on software version or feature level, etc.).

Appendix A contains more detailed descriptions of different footprint and capabilities management scenarios, but it is important to note that it is the ability of the dCDN to service each request in a functionally equivalent manner as the uCDN that is important, not the physical layout of resources through which it services the request. The aggregation of resource knowledge by the dCDN into a simple set of capabilities and their affective footprints, that is then advertised to the uCDN, enables efficient decision making at each delegation point in the CDN interconnection hierarchy.

It is assumed that an authoritative request router in each CDN will be responsible for aggregating and advertising capabilities information in a dCDN and/or receiving and aggregating capabilities

information in the uCDN. The CDNI Footprint & Capabilities Advertisement interface (FCI) along with the CDNI Request Routing Redirection interface (RI) [I-D.ietf-cdni-redirection] make up the CDNI Request Routing Interface. As there is no other centralized CDNI controller, the authoritative request router seems the most logical place for capabilities aggregation to occur, as it is the request router that needs such information to make delegation decisions. The protocol defined herein may be implemented as part of an entity other than an authoritative request router, but for the purposes of this discussion, the authoritative request router is assumed to be the centralized capabilities aggregation point.

Though there is an obvious need for the ability to exchange and update footprint and capability information in real-time, it is assumed that capabilities do not change very often. It is also assumed that the capabilities are not by themselves useful for making delegation decisions. Capability information is assumed to be input into business logic. It is the business logic which provides the algorithms for delegation decision making. The definition of business logic occurs outside the scope of CDNI and outside the timescale of footprint and capability advertisement [I-D.ietf-cdni-footprint-capabilities-semantics]. It may be the case that the business logic anticipates and reacts to changes in dCDN capabilities, however, it may also be the case that business logic is tailored through offline processes as dCDN capabilities change. The FCI is agnostic to the business processes employed by any given uCDN. The footprints and capabilities that are advertised over the FCI may be used by the uCDN at its discretion, to implement delegation rules. Setting proper defaults in the business logic should prevent any unwanted delegation from occurring when dCDN capabilities change, however, that is beyond the scope of this discussion.

1.1. Terminology

This document uses the terminology defined in section 1.1 of the CDNI Framework [RFC7336] document.

2. CDNI FCI Capability Advertisement

The FCI is implemented as an ALTO [RFC7285] Service. The ALTO protocol defines an HTTP-based transport through which ALTO service information may be retrieved using either a GET or POST method. The uCDN request router may at any time query the dCDN ALTO FCI Service for the full set of dCDN capability information. The uCDN may use a separate FCI Filter Service to retrieve a subset of the dCDN capability information.

[Ed.: Need to update this with ALTO asynchronous update support.]

[Ed.: Need to update this with ALTO incremental update support.]

2.1. CDNI FCI Capability Initialization

In lieu of any out-of-band pre-configured capability information, when the FCI is first brought up between a uCDN and a dCDN, the uCDN SHOULD assume that the dCDN has no CDNI capabilities. If an out-of-band capability baseline has been exchanged, the uCDN MAY use that information to initialize its capabilities database. In either case, the uCDN SHOULD verify the initial state of the dCDN (as a temporary outage may affect availability in the dCDN).

The dCDN MUST support sending its entire set of capabilities to the uCDN through the ALTO service interface

3. CDNI FCI Capabilities Service

As described in Requirement FCI-2, there is a basic set of capabilities that must be supported by the FCI for the uCDN to be able to determine if the dCDN is functionally able to handle a given request. [I-D.ietf-cdni-footprint-capabilities-semantics] lists mandatory capabilities types:

- Delivery Protocol
- Acquisition Protocol
- Redirection Mode
- CDNI Logging Capabilities
- CDNI Metadata Capabilities

To be consistent with the base ALTO service definitions, we use the JSON object definition notation as specified in the ALTO protocol [RFC7285].

3.1. CDNI FCI Map

3.1.1. Media Type

The media type of CDNI FCI Map is "application/alto-cdni-fcimap+json"

3.1.2. HTTP Method

A CDNI FCI Map resource is requested using the HTTP GET method.

3.1.3. Accept Input Parameters

None.

3.1.4. Capabilities

None.

3.1.5. Uses

None.

3.1.6. Response

The data component of a CDNI FCI Map resource is named "fcimap" which is a JSON object of type FCIMapData:

```
object {
   FCIMapData fcimap<0..*>;
} InfoResourceFCIMap : ResponseEntityBase;

object {
   FCICapability capabilities<1..*>;
} FCIMapData;

object {
   JSONString capability-type;
   JSONValue capability-value
   FCIFootprint footprints<0..*>;
} FCICapability;

object {
   JSONString footprint-type;
   JSONString footprint-value<1..*>;
} FCIFootprint;
```

The FCIMapData object contains a CDNI Payload Type [RFC7736] "ptype" which identifies the capability and a "value" object containing the associated Capability Advertisement Object (e.g., delivery-protocols, acquisition-protocols, or redirection-modes, as defined in [I-D.ietf-cdni-footprint-capabilities-semantics]). The FCIMapData object may also contain an optional list of FCIFootprint objects "footprints". The FCIFootprint object specifies a "footprint-type" (as defined by the CDNI Metadata Footprint Types registry, e.g., ipv4cidr, ipv6cidr, asn, or countrycode [I-D.ietf-cdni-metadata]) which identifies the contents and encoding of the individual footprint entries contained in the associated "footprint-value" array.

The "footprints" list MUST NOT contain multiple FCIFootprint objects of the same type. Footprint restriction information MAY be specified using multiple different footprint-types. If no footprint restriction list is specified (or an empty list is specified), it SHALL be understood that all footprint types MUST be reset to "global" coverage.

Note: Further optimization of the footprint object to provide quality information for a given footprint is certainly possible, however, it is not necessary for the basic interconnection of CDNs. The ability to transfer quality information in capabilities advertisements may be desirable and is noted here for completeness, however, the specifics of such mechanisms are outside the scope of this document.

Multiple FCIMapData objects with the same capability type are allowed within a given CDNI FCI Map response as long as the capability option footprint-value do not overlap, i.e., a given capability option value MUST NOT show up in multiple FCIMapData objects within a single CDNI FCI Map response. If multiple FCIMapData objects for a given capability type exist, those capability objects MUST have different footprint restrictions. Capability objects of a given capability type with identical footprint restrictions MUST be combined into a single capability object.

3.1.7. CDNI FCI Capabilities

3.1.7.1. Delivery Protocol

The delivery protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the delivery protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of delivery protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 627
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.DeliveryProtocol"
        "capability-value": {
          "delivery-protocols": [
            "https1.1"
          ]
        },
        "footprints": [
          { "footprint-type": "ipv4cidr",
            "footprint-value": [
              "10.1.0.0/16",
              "10.10.10.0/24"
            ]
          }
        ]
      }
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by IPv4 prefix).

A given protocol MUST NOT appear in multiple FCIMapData FCI.DeliveryProtocol object values.

3.1.7.2. Acquisition Protocol

The acquisition protocol refers to the protocol over which an end user (EU) has requested content. If a dCDN does not support the protocol requested by the client, then the dCDN is not a viable candidate for delegation.

Though the acquisition protocol is specified in the URI scheme (as defined in [RFC3986]) of the client request URL, protocol feature subsets or augmented protocol feature sets MAY be defined and SHOULD correspond with the protocols listed in the CDNI Metadata Protocol Types registry, e.g., http1.1 or https1.1 [I-D.ietf-cdni-metadata].

The following example shows two lists of acquisition protocols with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 620
Content-Type: application/alto-fcimap+json
{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "http1.1"
          ]
        }
      },
      { "capability-type": "FCI.AcquisitionProtocol"
        "capability-value": {
          "acquisition-protocols": [
            "https1.1"
          ]
        },
        "footprints": [
          { "footprint-type": "asn",
            "footprint-value": [
              "AS0",
              "AS65535"
            ]
          }
        ]
      }
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by Autonomous System number).

A given protocol MUST NOT appear in multiple FCIMapData FCI.AcquisitionProtocol value objects.

3.1.7.3. Redirection Mode

The redirection mode refers to the method(s) employed by request routers to perform request redirection. The CDNI framework [RFC7336] document describes four possible request routing modes:

- DNS iterative (DNS-I)
- DNS recursive (DNS-R)
- HTTP iterative (HTTP-I)
- HTTP recursive (HTTP-R)

[I-D.ietf-cdni-footprint-capabilities-semantics] defines the "CDNI Capabilities Redirection Modes" registry and the initial supported redirection mode values shown in parentheses above.

If a dCDN supports only a specific mode or subset of modes that does not overlap with the modes supported by the uCDN, then the dCDN might not be a viable candidate for delegation.

The following example shows two lists of redirection modes with different footprints.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 767
Content-Type: application/alto-fcimap+json

{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-I",
            "HTTP-I"
          ]
        }
      },
      { "capability-type": "FCI.RedirectionMode",
        "capability-value": {
          "redirection-modes": [
            "DNS-R",
            "HTTP-R"
          ]
        },
        "footprints": [
          { "footprint-type": "countrycode",
            "footprint-value": [
              "SE"
            ]
          },
          { "footprint-type": "ipv6cidr",
            "footprint-value": [
              "9876:5432::1/36"
            ]
          }
        ]
      }
    ]
  }
}
```

In the above example, iterative redirection is supported globally, while recursive redirection is only supported in a restricted

footprint (in this case, specified by both Country Code and IPv6 prefix).

A given mode MUST NOT appear in multiple FCIMapData FCI.RedirectionMode object values.

3.1.7.4. Logging Capabilities

[I-D.ietf-cdni-logging] describes the "cdni_http_request_v1" logging record-types and optional vs. mandatory-to-implement logging fields for that record-type. It also allows new logging record-types and logging fields to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain logging parameters which may affect billing agreements or legal requirements of the uCDN, then the dCDN is not a viable candidate for delegation.

3.1.7.4.1. CDNI Logging Capability Object Serialization

The following shows an example of CDNI Logging Capability Object Serialization, for a CDN that supports the optional Content Collection ID logging field (but not the optional Session ID logging field) for the "cdni_http_request_v1" record type.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1",
        "fields": [ "s-ccid" ]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

The next example shows the CDNI Logging Capability Object Serialization, for a CDN that supports all optional fields for the "cdni_http_request_v1" record type.

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Logging",
      "capability-value": {
        "record-type": "cdni_http_request_v1"
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

3.1.7.5. Metadata Capabilities

[I-D.ietf-cdni-metadata] describes GenericMetadata types which may be optional for a dCDN to implement, but which, if present, are mandatory-to-enforce. It also allows for new GenericMetadata to be defined which would be optional for a dCDN to implement.

If a dCDN does not support certain GenericMetadata types which are designated mandatory-to-enforce and may affect the correctness or security of the content being delivered, then the dCDN is not a viable candidate for delegation.

3.1.7.5.1. CDNI Metadata Capability Object Serialization

The following shows an example of CDNI Metadata Capability Object Serialization, for a CDN that supports only the SourceMetadata GenericMetadata type (i.e., it can acquire and deliver content, but cannot enforce and security policies, e.g., time, location, or protocol ACLs).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": ["MI.SourceMetadata"]
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

The next example shows the CDNI Metadata Capability Object Serialization, for a CDN that supports only structural metadata (i.e., it can parse metadata as a transit CDN, but cannot enforce security policies or deliver content).

```
{
  "capabilities": [
    {
      "capability-type": "FCI.Metadata",
      "capability-value": {
        "metadata": []
      },
      "footprints": [
        <Footprint objects>
      ]
    }
  ]
}
```

4. CDNI FCI Capabilities Filtering Service

4.1. Filtered CDNI FCI Map

4.1.1. Media Type

Since a Filtered CDNI FCI Map is still a CDNI FCI Map, it uses the media type defined for CDNI FCI Map (see Section 3.1.1).

4.1.2. HTTP Method

A Filtered CDNI FCI Map is requested using the HTTP POST method.

4.1.3. Accept Input Parameters

TBD.

4.1.4. Capabilities

None.

4.1.5. Uses

TBD.

4.1.6. Response

The format is the same as unfiltered CDNI FCI Map (see Section 3.1.6).

4.1.7. Example

TBD.

5. Footprint via ALTO Network Map

5.1. ALTO Network Maps

The ALTO Protocol offers an information service "ALTO map service" that provides information to ALTO clients in the form of Network Map and Cost Map [RFC7285]. As an alternative to the explicit definition of a CDNI Footprint Type (e.g., ipv4cidr, ipv6cidr, as, countrycode), a reference to an ALTO network map can be used to define an FCI footprint. To enable such referencing to ALTO network maps, a new CDNI Footprint Type "altonetworkmap" is defined (see also Section 6.2).

The first altonetworkmap entry must be a URI for accessing the ALTO server that hosts the ALTO network map (as defined in the ALTO protocol specification [RFC7285]). All subsequent altonetworkmap entries must be of type PIDName (as defined in [RFC7285], where the PIDName corresponds to a PID in the ALTO network map referenced by the preceding URI. Parsing and processing of an ALTO network map follows the ALTO protocol specification [RFC7285].

5.2. Example ALTO Network Map for CDNI FCI Footprint

An ALTO client can retrieve a network map of media type ’application/alto-networkmap+json’ under a given URI (e.g., ’http://alto.example.com/fcifootprint001’) with a GET request for a network map as specified in the ALTO protocol [RFC7285]. The following network map would convey to a uCDN that the given dCDN (which would provide the map) has three footprints called "south-france" and "germany", and provides the corresponding IPv4 address ranges for these footprints.

```
GET /networkmap HTTP/1.1
Host: http://alto.example.com/fcifootprint001
Accept: application/alto-networkmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 319
Content-Type: application/alto-networkmap+json

{
  "meta" : {
    "vtag": [
      {"resource-id": "my-eu-netmap",
       "tag": "1266506139"
      }
    ]
  },
  "network-map" : {
    "south-france" : {
      "ipv4" : [ "192.0.2.0/24", "198.51.100.0/25" ]
    },
    "germany" : {
      "ipv4" : [ "192.0.3.0/24"]
    }
  }
}
```

5.3. Example of ALTO Network Map Footprint in FCI Map

To reference an ALTO network map as an FCI footprint, set the footprint-type to "altonetworkmap", and set the first entry in the footprint-value to the URI of the ALTO server hosting the network map, followed by a list of PID names contained in the network map.

The following example shows two lists of delivery protocols (see Section 3.1.7.1), with the second having an ALTO network map footprint.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 618
Content-Type: application/alto-fcimap+json

{
  "meta" : {
  },
  "fcimap": {
    "capabilities": [
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
          "http1.1"
        ]
      },
      { "capability-type": "FCI.DeliveryProtocol",
        "capability-value": [
        "values": [
          "https1.1"
        ],
        "footprints": [
          { "footprint-type": "altonetworkmap",
            "footprint-value": [
              "http://alto.example.com/fcifootprint001",
              "germany",
              "south-france"
            ]
          }
        ]
      }
    ]
  }
}
```

In the above example, the HTTP/1.1 protocol is supported globally, while the HTTP/1.1 over TLS protocol is only supported in a restricted footprint (in this case, specified by an ALTO network map for Germany and Southern France).

6. IANA Considerations

6.1. ALTO Media Types

This document requests the registration of the following media types:

Type	Subtype
application	alto-cdni-fcimap+json
application	alto-cdni-fcimapfilter+json

6.1.1. ALTO CDNI FCI Map Media Type

Type name: application

Subtype name: alto-cdni-fcimap+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of [RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

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

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.1.2. ALTO CDNI FCI Map Filter Media Type

Type name: application

Subtype name: alto-cdni-fcimapfilter+json

Required parameters: none

Optional parameters: none

Encoding considerations:

Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations:

Security considerations relating to the generation and consumption of ALTO Protocol messages are discussed in Section 15 of

[RFC7285]. Additional security considerations for the CDNI Footprint & Capabilities Advertisement interface are discussed in Section 7.

Interoperability considerations:

[RFC7285] and RFCthis specify the format of conforming messages and the interpretation thereof. [RFC Editor: Please replace RFCthis with the published RFC number for this document.]

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

Applications that use this media type:

ALTO servers and ALTO clients either stand alone or are embedded within other applications.

Fragment identifier considerations: N/A

Additional information: N/A

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:

Kevin Ma <kevin.j.ma@ericsson.com>

Intended usage: LIMITED USE

Restrictions on usage:

This media type is intended only for use in CDNI Footprint & Capabilities Advertisement interface protocol message exchanges.

Author: IETF CDNI working group

Change controller: IETF CDNI working group

Provisional registration: no

6.2. CDNI Footprint Types

This document requests the following addition to the "CDNI Metadata Footprint Types" registry:

Footprint Type	Description	Specification
altonetworkmap	URI of an ALTO Server hosting an ALTO network map, followed by a list of PID-names	RFCthis

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

7. Security Considerations

There are a number of security concerns associated with the FCI. The FCI essentially provides configuration information which the uCDN uses to make request routing decisions. Injection of fake capability advertisement messages or the interception and discard of real capability advertisement messages may be used for denial of service (e.g., by falsely advertising or deleting capabilities or preventing capability advertisements from reaching the uCDN). FCI messages may also be monitored to detect when CDN performance degrades or outages occur. Such information may be considered private by the dCDN.

dCDN capability advertisements MUST be authenticated by the uCDN to prevent unauthorized capability injection. uCDN FCI servers MUST be authenticated by the dCDN to prevent unauthorized interception of ALTO messages. Encryption MUST be used to ensure confidentiality of the dCDN’s private messages.

7.1. Securing the CDNI Footprint & Capabilities Advertisement interface

An implementation of the CDNI Footprint & Capabilities Advertisement interface MUST support TLS transport as per [RFC2818] and [RFC7230]. The use of TLS for transport of the CDNI metadata interface messages allows:

- The dCDN and uCDN to authenticate each other.

and, once they have mutually authenticated each other, it allows:

- The dCDN and uCDN to authorize each other (to ensure they are transmitting/receiving CDNI FCI messages from an authorized CDN);

- CDNI FCI messages to be transmitted with confidentiality; and
- The integrity of the CDNI FCI messages to be protected during the exchange.

In an environment where any such protection is required, TLS MUST be used (including authentication of the remote end) by the server-side (uCDN) and the client-side (dCDN) of the CDNI Footprint & Capabilities Advertisement interface unless alternate methods are used for ensuring the confidentiality of the information in the CDNI FCI messages (such as setting up an IPsec tunnel between the two CDNs or using a physically secured internal network between two CDNs that are owned by the same corporate entity).

When TLS is used, the general TLS usage guidance in [RFC7525] MUST be followed.

8. Acknowledgements

The authors would like to thank Jon Peterson, Ray van Brandenburg, Gilles Bertrand, and Scott Wainner for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

9. References

9.1. Normative References

[I-D.ietf-cdni-footprint-capabilities-semantics]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantics-16 (work in progress), April 2016.

[I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-25 (work in progress), April 2016.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-15 (work in progress), April 2016.

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

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

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <http://www.rfc-editor.org/info/rfc7285>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

9.2. Informative References

[I-D.ietf-cdni-redirection]
Niven-Jenkins, B. and R. Brandenburg, "Request Routing Redirection interface for CDN Interconnection", draft-ietf-cdni-redirection-18 (work in progress), April 2016.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[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, <http://www.rfc-editor.org/info/rfc3986>.

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

[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, <http://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, <http://www.rfc-editor.org/info/rfc7337>.

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

Appendix A. Capability Aggregation

The following sections show examples of three aggregation scenarios. In each case, CDN-U is the ultimate uCDN and CDN-P is the penultimate CDN which must perform capabilities aggregation.

A.1. Downstream CDN Aggregation

Figure A1 shows five organizationally different CDNs: CDN-U, CDN-P, and CDNS A, B, and C, the dCDNs of CDN-P which are being aggregated. Given the setup shown in Figure A1, we can construct a number of use cases, based on the coverage areas of each dCDN (i.e., CDNs P, A, B, and C). Note: In all cases, the reachability of the uCDN (i.e., CDN-U) is a don’t care as it is assumed that the uCDN knows its own coverage area and is likely to favor itself in most situations, and if it has decided that it needs to delegate to a dCDN, then the only relevant question is if the dCDN can handle the request.

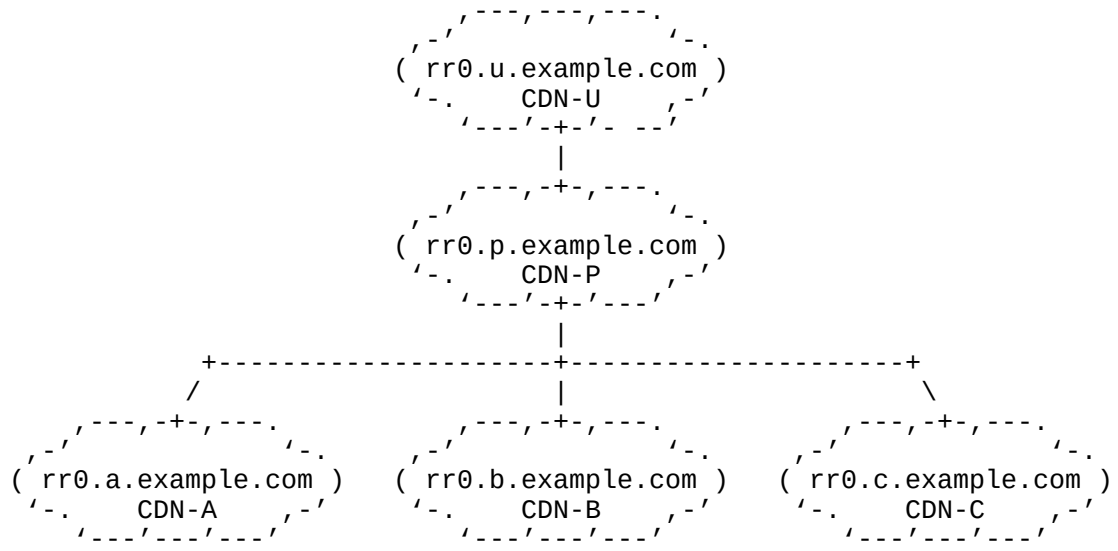


Figure A1: CDNI dCDN Request Router Aggregation

- None of the four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, each CDN is likely to advertise footprint information with its capabilities, specifying its reachability. When CDN-P advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and CDNs A, B, and C. Note: CDN-P MAY exclude any dCDN, and consequently its footprint, per its own internal aggregation decision criteria.

- All four dCDNs (CDNs P, A, B, and C) have global reachability. In this case, none of the CDNs is likely to advertise any footprint information as none have any footprint restrictions. When CDN-P advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.

- Some of the four dCDNs (CDNs P, A, B, and C) have global reachability and some do not. In this case, even though some dCDNs do not have global reachability, the aggregate of some dCDNs having global reachability and some not should still be global reachability (for the given capability). When CDN-P advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one dCDN has global reach as being supported with global reachability. It is up to the CDN-P request router to properly select a dCDN to process individual client requests and not choose a dCDN whose restricted footprint makes it unsuitable for delivering the requested content.

A.2. Internal Request Router Aggregation

Figure A2 shows CDN-U and CDN-P where CDN-P internally has four request routers: the authoritative request router rr0, and three other request routers rr1, rr2, and rr3. The use of multiple request routers may be used to distribute request routing load across resources, possibly in different geographic regions covered by CDN-P. Similar to Figure A1, the setup shown in Figure A2 requires the authoritative request router rr0 in CDN-P to aggregate capabilities information from downstream request routers rr1, rr2, and rr3. The primary difference between the scenario is that the request routers in Figure A2 are logically within the same CDN-P organization. The same reachability scenarios apply to Figure A2 as with Figure A1.

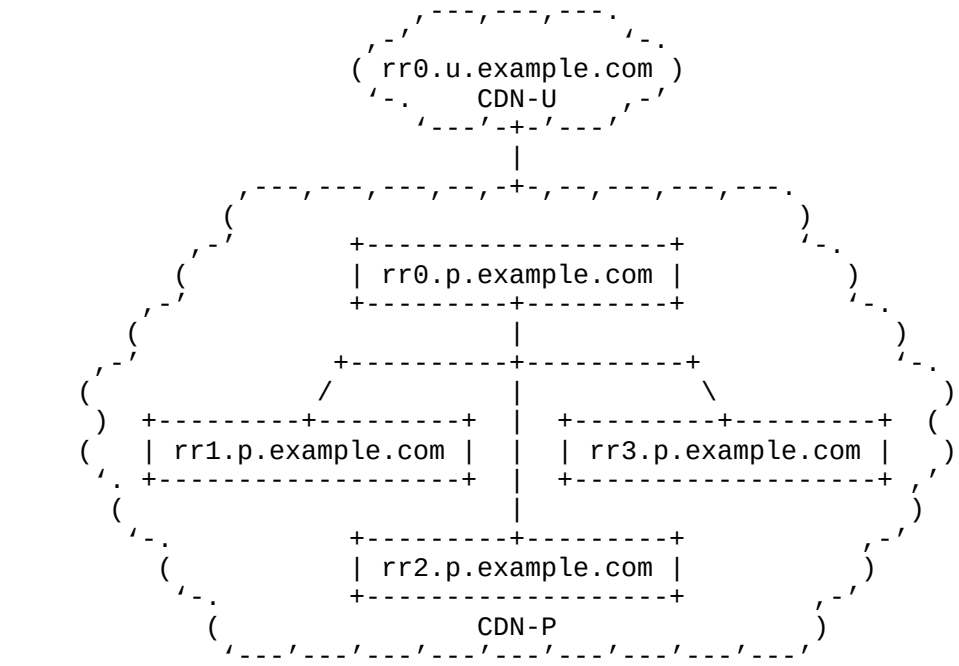


Figure A2: Local CDN Request Router Aggregation

- None of the four CDN-P request routers have global reachability. In this case, each request router is likely to advertise footprint information with its capabilities, specifying its reachability. When rr0 advertises capabilities to CDN-U, it may advertise the aggregate footprint of itself and rr1, rr2, and rr3.

- All four CDN-P request routers have global reachability. In this case, none of the request routers is likely to advertise any

footprint information as none has any footprint restrictions. When rr0 advertises capabilities to CDN-U, the aggregate of all global reachability is global reachability.

- o Some of the four CDN-P request routers have global reachability and some do not. In this case, even though some request routers do not have global reachability, the aggregate of some request routers having global reachability and some not should still be global reachability (for the given capability). When rr0 advertises capabilities to CDN-U, CDN-P may advertise capabilities for which at least one request router has global reach as being supported with global reachability. It is up to the authoritative request router rr0 to properly select from the other request routers for any given request, and not choose a request router whose restricted footprint makes it unsuitable for delivering the requested content.

A.3. Internal Capability Aggregation

Figure A3 shows CDN-U and CDN-P where the delivery network of CDN-P is segregated by delivery protocol (e.g., RTSP, HTTP, and RTMP). Figure A3 differs from Figures A1 and A2 in that request router rr0 of CDN-P is not aggregating the capabilities advertisements of multiple other downstream request routers, but rather it is managing the disparate capabilities across resources within its own local CDN. Though not every delivery node has the same protocol capabilities, the aggregate delivery protocol capabilities advertised by CDN-A may include all delivery protocols. Note, Figure A3 should not be construed to imply anything about the coverage areas for each delivery protocol. They may all support the same delivery footprint, or they may have different delivery footprints. It is the responsibility of the request router rr0 to properly assign protocol-appropriate delivery nodes to individual content requests. If certain protocols have limited reachability, CDN-P may advertise footprint restrictions for each protocol.

It should be noted that though the delivery protocol capability was selected for this example, the concept of internal capability aggregation applies to all capabilities as discussed below.

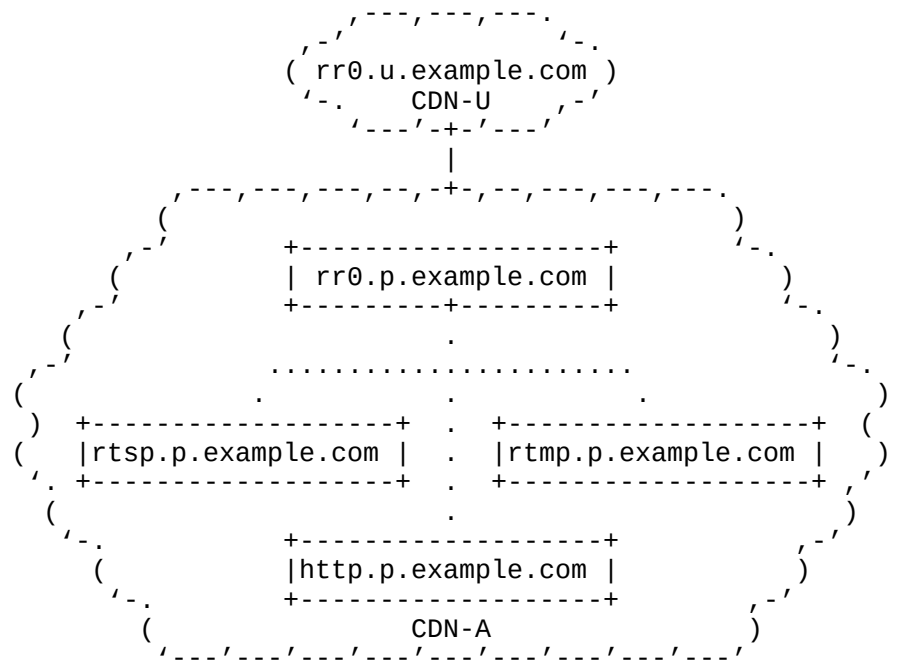


Figure A3: Local CDN Capability Segregation

Another situation in which physical footprint may not matter in an aggregated view has to do with feature support (e.g., new CDNI metadata features or new redirection modes). Situations often arise when phased roll-out of software upgrades, or staging network segregation result in only certain portions of a CDN's resources supporting the new feature set. The dCDN has a few options in this case:

- Enforce atomic update: The dCDN does not advertise support for the new capability until all resources have been upgraded to support the new capability.

- Transparent segregation: The dCDN advertises support for the new capability, and when requests are received that require the new capability, the dCDN request router properly selects a resource which supports that capability.

- Advertised segregation: The dCDN advertises support for the new capability with a footprint restriction allowing the uCDN to make delegation decisions based on the dCDN's limit support.

The level of aggregation employed by the dCDN is likely to vary as business relationships dictate, however, the FCI should support all possible modes of operation.

Authors' Addresses

Kevin J. Ma
Ericsson
43 Nagog Park
Acton, MA 01720
USA

Phone: +1 978-844-5100
Email: kevin.j.ma@ericsson.com

Jan Seedorf
NEC
Kurfuerstenanlage 36
Heidelberg 69115
Germany

Phone: +49 6221 4342 221
Fax: +49 6221 4342 155
Email: seedorf@neclab.eu

CDNI
Internet-Draft
Intended status: Informational
Expires: September 6, 2015

J. Seedorf
NEC
Y. Yang
Yale
J. Peterson
Neustar

March 5, 2015

CDNI Footprint and Capabilities Advertisement using ALTO
draft-seedorf-cdni-request-routing-alto-08

Abstract

Network Service Providers (NSPs) are currently considering to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. The necessary interfaces for inter-connecting CDNs are currently being defined in the Content Delivery Networks Interconnection (CDNI) WG. This document focuses on the CDNI Footprint & Capabilities Advertisement interface (FCI). Specifically, this document specifies a new Application Layer Traffic Optimization (ALTO) service to facilitate Footprint & Capabilities Advertisement in a CDNI context.

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 http://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 September 6, 2015.

Copyright Notice

Table of Contents

1. Introduction

Many Network Service Providers (NSPs) are currently considering or have already started to deploy Content Delivery Networks (CDNs) within their networks. As a consequence of this development, there is a need for interconnecting these local CDNs. Content Delivery Networks Interconnection (CDNI) has the goal of standardizing protocols to enable such interconnection of CDNs [RFC6707].

The CDNI problem statement [RFC6707] envisions four interfaces to be standardized within the IETF for CDN interconnection:

- CDNI Request Routing Interface

- CDNI Metadata Interface
- CDNI Logging Interface
- CDNI Control Interface

This document focuses solely on the CDNI Request Routing Interface, which can be further divided into two interfaces (see [RFC6707] for a detailed description): the CDNI Request Routing Redirection interface (RI), and the CDNI Footprint & Capabilities Advertisement interface (FCI). This document specifies a new Application Layer Traffic Optimization (ALTO) [RFC7285] service called 'CDNI Footprint & Capabilities Advertisement Service'. This service is used to transport CDNI FCI JSON objects, which are defined in a separate document [I-D.ma-cdni-capabilities]. An abstraction for managing individual CDNI capabilities in an opaque manner is defined as 'FCIBase' object in [I-D.ietf-cdni-footprint-capabilities-semantics].

Throughout this document, we use the terminology for CDNI defined in [RFC6707].

2. ALTO within CDNI Request Routing

The main purpose of the CDNI Request Routing Interface is described in [RFC6707] as follows: "The CDNI Request Routing interface enables a Request Routing function in an Upstream CDN to query a Request Routing function in a Downstream CDN to determine if the Downstream CDN is able (and willing) to accept the delegated Content Request. It also allows the Downstream CDN to control what should be returned to the User Agent in the redirection message by the upstream Request Routing function." On a high level, the scope of the CDNI Request Routing Interface therefore contains two main tasks:

- A) Determining if the downstream CDN is willing to accept a delegated content request
- B) Redirecting the content request coming from an upstream CDN to the proper entry point or entity in the downstream CDN

More precisely, in [RFC7336] the request routing interface is broadly divided into two functionalities:

- 1) the asynchronous advertisement of footprint and capabilities by a dCDN that allows a uCDN to decide whether to redirect particular user requests to that dCDN (the CDNI FCI)
- 2) the synchronous operation of actually redirecting a user request (the CDNI RI)

Application Layer Traffic Optimization (ALTO) [RFC7285] is an approach for guiding the resource provider selection process in distributed applications that can choose among several candidate resources providers to retrieve a given resource. By conveying network layer (topology) information, an ALTO server can provide important information to "guide" the resource provider selection process in distributed applications. Usually, it is assumed that an ALTO server conveys information these applications cannot measure themselves [RFC5693].

Originally, ALTO was motivated by the huge amount of cross-ISP traffic generated by P2P applications [RFC5693]. Recently, however, ALTO is also being considered for improving the request routing in CDNs [I-D.jenkins-alto-cdn-use-cases]. In this context, it has also been proposed to use ALTO for selecting an entry-point in a downstream NSP's network (see section 3.4 "CDN delivering Over-The-Top of a NSP's network" in [I-D.jenkins-alto-cdn-use-cases]). Also, the CDNI problem statement explicitly mentions ALTO as a candidate protocol for "algorithms for selection of CDN or Surrogate by Request-Routing systems" [RFC6707].

3. Assumptions and High-Level Design Considerations

In this section we list some assumptions and design issues to be considered when using ALTO for the CDNI Footprint and Capabilities Advertisement interface.

3.1. General Assumptions and Considerations

Below we list some general assumptions and considerations:

- As explicitly being out-of-scope for CDNI [RFC6707], it is assumed that ingestion of content or acquiring content across CDNs is not part of request routing as considered within CDNI standardization work. The focus of using ALTO (as considered in this document) is hence on request routing only, assuming that the content (desired by the end user) is available in the downstream CDN (or can be aquired by the downstream CDN by some means).

- Federation Model: "Footprint and Capabilities Advertisement" and in general CDN request routing depends on the federation model among the CDN providers. Designing a suitable solution thus depends on whether a solution is needed for different settings, where CDNs consist of both NSP CDNs (serving individual ASes) and general, traditional CDNs (such as Akamai). We assume that CDNI is not designed for a setting where only NSP CDNs each serve a single AS only.

- In this document, it is assumed that the upstream CDN (uCDN) makes the decision on selecting a downstream CDN, based on information that each downstream CDN has made available to the upstream CDN. Further, we assume that in principle more than one dCDN may be suitable for a given end-user request (i.e. different dCDNs may claim "overlapping" footprints). The uCDN hence potentially has to select among several candidate downstream CDNs for a given end user request.

- It is not clear what kind(s) of business, contract, and operational relationships two peering CDNs may form. For the Internet, we see provider-customer and peering as two main relations; providers may use different charging models (e.g., 95-percentile, total volume) and may provide different SLAs. Given such unknown characteristics of CDN peering business agreements, we should design the protocol to support as much diverse potential business and operational models as possible.

3.2. Semantics for Footprint/Capabilities Advertisment

The CDNI document on "Footprint and Capabilities Semantics" [I-D.ietf-cdni-footprint-capabilities-semantics] defines the semantics for the CDNI FCI. It thus provides guidance on what Footprint and Capabilities mean in a CDNI context and how a protocol solution should in principle look like. Here we briefly summarize the key points of the semantics of Footprint and Capabilities (for a detailed discussion, the reader is referred to [I-D.ietf-cdni-footprint-capabilities-semantics]):

- Often, footprint and capabilities are tied together and cannot be interpreted independently from each other. In such cases, i.e. where capabilities must be expressed on a per footprint basis, it may be beneficial to combine footprint and capabilities advertisement.

- Given that a large part of Footprint and Capabilities Advertisement will actually happen in contractual agreements, the semantics of CDNI Footprint and Capabilities advertisement refer to answering the following question: what exactly still needs to be advertised by the CDNI FCI? For instance, updates about temporal failures of part of a footprint can be useful information to convey via the CDNI request routing interface. Such information would provide updates on information previously agreed in contracts between the participating CDNs. In other words, the CDNI FCI is a means for a dCDN to provide changes/updates regarding a footprint and/or capabilities it has prior agreed to serve in a contract with a uCDN.

- It seems clear that "coverage/reachability" types of footprint must be supported within CDNI. The following such types of footprint are mandatory and must be supported by the CDNI FCI:

  * List of ISO Country Codes

  * List of AS numbers

  * Set of IP-prefixes

  A 'set of IP-prefixes' must be able to contain full IP addresses, i.e., a /32 for IPv4 and a /128 for IPv6, and also IP prefixes with an arbitrary prefix length. There must also be support for multiple IP address versions, i.e., IPv4 and IPv6, in such a footprint.

- For all of these mandatory-to-implement footprint types, footprints can be viewed as constraints for delegating requests to a dCDN: A dCDN footprint advertisement tells the uCDN the limitations for delegating a request to the dCDN. For IP prefixes or ASN(s), the footprint signals to the uCDN that it should consider the dCDN a candidate only if the IP address of the request routing source falls within the prefix set (or ASN, respectively). The CDNI specifications do not define how a given uCDN determines what address ranges are in a particular ASN. Similarly, for country codes a uCDN should only consider the dCDN a candidate if it covers the country of the request routing source. The CDNI specifications do not define how a given uCDN determines the country of the request routing source. Multiple footprint constraints are additive, i.e. the advertisement of different types of footprint narrows the dCDN candidacy cumulatively.

- The following capabilities seem useful as 'base' capabilities, i.e. ones that are needed in any case and therefore constitute mandatory capabilities to be supported by the CDNI FCI:

  * Delivery Protocol (e.g., HTTP vs. RTMP)

  * Acquisition Protocol (for aquiring content from a uCDN)

  * Redirection Mode (e.g., DNS Redirection vs. HTTP Redirection as discussed in [RFC7336])

  * Capabilities related to CDNI Logging (e.g., supported logging mechanisms)

* Capabilities related to CDNI Metadata (e.g., authorization algorithms or support for proprietary vendor metadata)

3.3. Advantages of using ALTO as the CDNI FCI protocol

The following reasons make ALTO a suitable candidate protocol for downstream CDN selection as part of CDNI request routing and in particular for an FCI protocol:

- CDN request routing is done at the application layer. ALTO is a protocol specifically designed to improve application layer traffic (and application layer connections among hosts on the Internet) by providing additonal information to applications that these applications could not easily retrieve themselves. For CDNI, this is exactly the case: a uCDN wants to improve application layer CDN request routing by using dedicated information (provided by a dCDN) that the uCDN could not easily obtain otherwise.

- The semantics of an ALTO network map are an exact match for the needed information to convey a footprint by a downstream CDN, in particular if such a footprint is being expressed by IP-prefix ranges.

- Security: ALTO maps can be signed and hence provide inherent integrity protection (see Section 6)

- RESTful-Design: The ALTO protocol has undergone extensive revisions in order to provide a RESTful design regarding the client-server interaction specified by the protocol. A CDNI FCI interface based on ALTO would inherit this RESTful design.

- Error-handling: The ALTO protocol has undergone extensive revisions in order to provide sophisticated error-handling, inparticular regarding unexpected cases. A CDNI FCI interface based on ALTO would inherit this thought-through and mature error-handling.

- Filtered network map: The ALTO Map Filtering Service (see [RFC7285] for details) would allow a uCDN to query only for parts of an ALTO map.

3.4. Selection of a Downstream CDN with ALTO

Under the considerations stated in Section 3, ALTO can help the upstream CDN provider to select a proper downstream CDN provider for a given end user request as follows: Each downstream CDN provider

hosts an ALTO server which provides ALTO services which convey CDNI FCI information to an ALTO client at the upstream CDN provider.

4. CDNI FCI ALTO Service

The ALTO protocol is based on an ALTO Information Service Framework which consists of several services, where all ALTO services are 'provided through a common transport protocol, messaging structure and encoding, and transaction model' [RFC7285]. The ALTO protocol specification [RFC7285] defines several such services, e.g. the ALTO map service.

This document defines a new ALTO Service called 'CDNI Footprint & Capabilities Advertisement Service' which conveys JSON objects of media type 'application/alto-fcimap+json'. This media type and JSON object format is defined in [I-D.ma-cdni-capabilities]; this document specifies how to transport such JSON objects via the ALTO protocol with the ALTO 'CDNI Footprint & Capabilities Advertisement Service'. An abstraction for managing individual CDNI capabilities in an opaque manner is defined as 'FCIBase' object in [I-D.ietf-cdni-footprint-capabilities-semantics].

4.1. Server Response Encoding

4.1.1. CDNI FCI Map

The media type of the CDNI FCI Map is 'application/alto-cdni-fcimap+json'. The HTTP Method, Accept Input Parameters, Capabilities, Uses, and Response of the CDNI FCI Map are specified in [I-D.ma-cdni-capabilities].

4.1.2. Meta Information

The 'meta' field of a FCIMapData response MUST include 'vtag', which is an ALTO Version Tag of the retrieved FCIMapData according to [RFC7285] (Section 10.3.). It thus contains a 'resource-id' attribute, and a 'tag' is an identifier string.

4.1.3. Data Information

The data component of a CDNI FCI Map resource is named 'fcimap' which is a JSON object of type FCIMapData. This JSON object of type FCIMapData is derived from ResponseEntityBase as specified in the ALTO protocol [RFC7285] (Section 8.4.) and specified in [I-D.ma-cdni-capabilities].

4.2. Protocol Errors

Protocol errors are handled as specified in the ALTO protocol [RFC7285] (Section 8.5.).

4.3. Example

The following example shows an CDNI FCI Map as in [I-D.ma-cdni-capabilities], however with meta-information as defined in Section 4.1.2 of this document.

```
GET /fcimap HTTP/1.1
Host: alto.example.com
Accept: application/alto-fcimap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 439
Content-Type: application/alto-fcimap+json
{
  "meta" : {
    "vtag": {
      "resource-id": "my-default-fcimap",
       "tag": "da65eca2eb7a10ce8b059740b0b2e3f8eb1d4785"
    }
  },
  "fcimap": [
    { "name": "delivery_protocol",
      "values": [
        "HTTP",
        "RTSP",
        "MMS"
      ]
    },
    { "name": "delivery_protocol",
      "values": [
        "RTMP",
        "HTTPS"
      ],
      "footprint": [
        { "type": "IPv4CIDR",
          "values": [
            "10.1.0.0/16",
            "10.10.10.0/24"
          ]
        }
      ]
    }
  ]
}
```

5. Useful ALTO extensions for CDNI Request Routing

It is envisioned that yet-to-be-defined ALTO extensions will be standardized that make the ALTO protocol more suitable and useful for applications other than the originally considered P2P use case [I-D.marocco-alto-next]. Some of these extensions to the ALTO protocol would be useful for ALTO to be used as a protocol within CDNI request routing, and in particular within the "Footprint and

Capabilities Advertisment" part of the CDNI request routing interface.

The following proposed extensions to ALTO would be beneficial to facilitate CDNI request routing with ALTO as outlined in Section 3.4:

- Server-initiated Notifications and Incremental Updates: In case the footprint or the capabilities of a downstream CDN change abruptly (i.e. unexpectedly from the perspective of an upstream CDN), server initiated notifications would enable a dCDN to directly inform an upstream CDN about such changes. Consider the case where - due to failure - part of the footprint of the dCDN is not functioning, i.e. the CDN cannot serve content to such clients with reasonable QoS. Without server-initiated notifications, the uCDN might still use a very recent network and cost map from dCDN, and therefore redirect request to dCDN which it cannot serve. Similarly, the possibility for incremental updates would enable efficient conveyance of the aforementioned (or similar) status changes by the dCDN to the uCDN. A proposal for server-initiated ALTO updates can be found in [I-D.marocco-alto-ws]. A discussion of incremental ALTO updates can be found in [I-D.schwan-alto-incr-updates].

- Content Availability on Hosts: A dCDN might want to express CDN capabilties in terms of certain content types (e.g. codecs/ formats, or content from certain content providers). A new endpoint property for ALTO that would be able to express such "content availability" would enable a dCDN to make available such information to an upstream CDN. This would enable a uCDN to determine if a given dCDN actually has the capabilities for a given request with respect to the type of content requested.

- Resource Availability on Hosts or Links: The capabilities on links (e.g. maximum bandwidth) or caches (e.g. average load) might be useful information for an upstream CDN for optimized dowmstream CDN selection. For instance, if a uCDN receives a streaming request for content with a certain bitrate, it needs to know if it is likely that a dCDN can fulfill such stringent application-level requirements (i.e. can be expected to have enough consistent bandwidth) before it redirects the request. In general, if ALTO could convey such information via new endpoint properties, it would enable more sophisticated means for downstream CDN selection with ALTO.

6. Security Considerations

One important security consideration is the proper authentication of advertisement information provided by a downstream CDN. The ALTO protocol provides a specification for a signature of ALTO information (see 8.2.2. of [RFC7285]. ALTO thus provides a proper means for protecting the integrity of FCI information.

More Security Considerations will be discussed in a future version of this document.

7. Acknowledgements

The authors would like to thank Kevin Ma, Daryl Malas, and Matt Caulfield for their timely reviews and invaluable comments.

Jan Seedorf is partially supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

8. References

8.1. Normative References

[RFC5693] Seedorf, J. and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement", RFC 5693, October 2009.

[RFC6707] Niven-Jenkins, B., Le Faucheur, F., and N. Bitar, "Content Distribution Network Interconnection (CDNI) Problem Statement", RFC 6707, September 2012.

[RFC6770] Bertrand, G., Stephan, E., Burbridge, T., Eardley, P., Ma, K., and G. Watson, "Use Cases for Content Delivery Network Interconnection", RFC 6770, November 2012.

[RFC7285] Alimi, R., Penno, R., Yang, Y., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, September 2014.

[RFC7336] Peterson, L., Davie, B., and R. van Brandenburg, "Framework for Content Distribution Network Interconnection (CDNI)", RFC 7336, August 2014.

[RFC7337] Leung, K. and Y. Lee, "Content Distribution Network Interconnection (CDNI) Requirements", RFC 7337, August 2014.

8.2. Informative References

[I-D.peterson-CDNI-strawman] Peterson, L. and J. Hartman, "Content Distribution Network Interconnection (CDNI) Problem Statement", draft-peterson-CDNI-strawman-01 (work in progress), May 2011.

[I-D.marocco-alto-next] Marocco, E. and V. Gurbani, "Extending the Application-Layer Traffic Optimization (ALTO) Protocol", draft-marocco-alto-next-00 (work in progress), January 2012.

[I-D.marocco-alto-ws] Marocco, E. and J. Seedorf, "WebSocket-based server-to-client notifications for the Application-Layer Traffic Optimization (ALTO) Protocol", draft-marocco-alto-ws-02 (work in progress), February 2014.

[I-D.schwan-alto-incr-updates] Schwan, N. and B. Roome, "ALTO Incremental Updates", draft-schwan-alto-incr-updates-02 (work in progress), July 2012.

[I-D.jenkins-alto-cdn-use-cases] Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-03 (work in progress), June 2012.

[I-D.ma-cdni-capabilities] Ma, K. and J. Seedorf, "CDNI Footprint & Capabilities Advertisement Interface", draft-ma-cdni-capabilities-06 (work in progress), June 2014.

[I-D.liu-cdni-cost] Liu, H., "A Cost Perspective on Using Multiple CDNs", draft-liu-cdni-cost-00 (work in progress), October 2011.

[I-D.ietf-cdni-metadata]
Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "CDN Interconnection Metadata", draft-ietf-cdni-metadata-09 (work in progress), March 2015.

[I-D.ietf-cdni-logging]
Faucheur, F., Bertrand, G., Oprescu, I., and R. Peterkofsky, "CDNI Logging Interface", draft-ietf-cdni-logging-15 (work in progress), February 2015.

[I-D.ietf-cdni-footprint-capabilities-semantics]
Seedorf, J., Peterson, J., Previdi, S., Brandenburg, R., and K. Ma, "CDNI Request Routing: Footprint and Capabilities Semantics", draft-ietf-cdni-footprint-capabilities-semantics-05 (work in progress), March 2015.

Authors' Addresses

Jan Seedorf
NEC Laboratories Europe, NEC Europe Ltd.
Kurfuersten-Anlage 36
Heidelberg 69115
Germany

Phone: +49 (0) 6221 4342 221
Email: jan.seedorf@neclab.eu
URI: http://www.neclab.eu

Y.R. Yang
Yale University
51 Prospect Street
New Haven 06511
USA

Email: yry@cs.yale.edu
URI: http://www.cs.yale.edu/~yry/

Jon Peterson
NeuStar
1800 Sutter St Suite 570
Concord CA 94520
USA

Email: jon.peterson@neustar.biz