                 'Out-Of-Band' Content Coding for HTTP
                   draft-reschke-http-oob-encoding-07

Abstract

   This document describes an Hypertext Transfer Protocol (HTTP) content
   coding that can be used to describe the location of a secondary
   resource that contains the payload.

Editorial Note (To be removed by RFC Editor before publication)

   Distribution of this document is unlimited. Although this is not a
   work item of the HTTPbis Working Group, comments should be sent to
   the Hypertext Transfer Protocol (HTTP) mailing list at
   ietf-http-wg@w3.org [1], which may be joined by sending a message
   with subject "subscribe" to ietf-http-wg-request@w3.org [2].

   Discussions of the HTTPbis Working Group are archived at
   <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

   XML versions, latest edits, and issue tracking for this document are
   available from
   <https://github.com/EricssonResearch/Blind-Cache-Drafts> and
   <http://greenbytes.de/tech/webdav/#draft-reschke-http-oob-encoding>.

   The changes in this draft are summarized in Appendix C.7.

Status of This Memo

This Internet-Draft will expire on January 6, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document describes an Hypertext Transfer Protocol (HTTP) content coding (Section 3.1.2.1 of [RFC7231]) that can be used to describe the location of a secondary resource that contains the payload.

The primary use case for this content coding is to enable origin servers to securely delegate the delivery of content to a secondary server that might be "closer" to the client (with respect to network topology) and/or able to cache content ([SCD]), leveraging content encryption ([ENCRYPTENC]).

2. Notational Conventions

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 terminology used in the base HTTP specifications, namely Section 2 of [RFC7230] and Section 3 of [RFC7231].

3. 'Out-Of-Band' Content Coding

3.1. Overview

The 'Out-Of-Band' content coding is used to direct the recipient to retrieve the actual message representation (Section 3 of [RFC7231]) from a secondary resource, such as a public cache:

1. Client performs a request

2. Received response specifies the 'out-of-band' content coding; the payload of the response contains additional meta data, plus the location of the secondary resource

3. Client performs GET request on secondary resource (usually again via HTTP(s))

4. Secondary server provides payload

5. Client combines above representation with additional representation metadata obtained from the primary resource

```
       Client                    Secondary Server         Origin Server

          sends GET request with Accept-Encoding: out-of-band
      (1) |----------------------------------------------------------\
                         status 200 and Content-Coding: out-of-band |
      (2) <----------------------------------------------------------/

          GET to secondary server
      (3) |--------------------------\
                              payload |
      (4) <--------------------------/

      (5)
         Client and combines payload received in (4)
         with metadata received in (2).
```

3.2. Definitions

The name of the content coding is "out-of-band".

The payload format uses JavaScript Object Notation (JSON, [RFC7159]),
describing an object describing secondary resources; currently only
defining one member:

'sr' A REQUIRED string array containing at least one URI reference
      (Section 4.1 of [RFC3986]) of a secondary resource (URI references
      that are relative references are resolved against the URI of the
      primary resource).

[[pext: This payload might be too simple in that there's no simple
way to annotate the secondary resources.]]

The payload format uses an array so that the origin server can
specify multiple secondary resources. The ordering within the array
reflects the origin server's preference (if any), with the most
preferred secondary resource location being first. Clients receiving
a response containing multiple URIs are free to choose which of these
to use.

In some cases, the origin server might want to specify a "fallback
URI"; identifying a secondary resource served by the origin server
itself, but otherwise equivalent "regular" secondary resources. Any
secondary resource hosted by the origin server can be considered to
be a "fallback"; origin servers will usually list them last in the
"sr" array so that they only will be used by clients when there is no
other choice.

New specifications can define new OPTIONAL header fields, thus

clients MUST ignore unknown fields. Extension specifications will
have to update this specification. [[anchor3: or we define a
registry]]

3.3. Processing Steps

Upon receipt of an 'out-of-band' encoded response, a client first
needs to obtain the secondary resource's presentation. This is done
using an HTTP GET request (independently of the original request
method).

In order to prevent any leakage of information, the GET request for
the secondary resource MUST only contain information provided by the
origin server or the secondary server itself, namely HTTP
authentication credentials ([RFC7235]) and cookies ([RFC6265]).

Furthermore, the request MUST include an "Origin" header field
indicating the origin of the original resource ([RFC6454], Section
7). The secondary server MUST verify that the specified origin is
authorized to retrieve the given payload (or otherwise return an
appropriate 4xx status code).

After receipt of the secondary resource's payload, the client then
reconstructs the original message by:

1. Unwrapping the encapsulated HTTP message by removing any transfer
    and content codings.

2. Replacing/setting any response header fields from the primary
    response except for framing-related information such as Content-
    Length, Transfer-Encoding and Content-Encoding.

If the client is unable to retrieve the secondary resource's
representation (host can't be reached, non 2xx response status code,
payload failing integrity check, etc.), it can choose an alternate
secondary resource (if specified), try the fallback URI (if given),
or simply retry the request to the origin server without including
'out-of-band' in the Accept-Encoding request header field. In the
latter case, it can be useful to inform the origin server about what
problems were encountered when trying to access the secondary
resource; see Section 3.4 for details.

Note that although this mechanism causes the inclusion of external
content, it will not affect the application-level security properties
of the reconstructed message, such as its web origin ([RFC6454]).

The cacheability of the response for the secondary resource does not
affect the cacheability of the reconstructed response message, which

is the same as for the origin server's response.

Use of the 'out-of-band' coding is similar to HTTP redirects ([RFC7231], Section 6.4) in that it can lead to cycles. Unless with HTTP redirects, the client however is in full control: it does not need to advertise support for the 'out-of-band' coding in requests for secondary resources. Alternatively, it can protect itself just like for HTTP redirects -- by limiting the number of indirections it supports.

Note that because the server's response depends on the request's Accept-Encoding header field, the response usually will need to be declared to vary on that. See Section 7.1.4 of [RFC7231] and Section 2.3 of [RFC7232] for details.

3.4. Problem Reporting

When the client fails to obtain the secondary resource, it can be useful to inform the origin server about the condition. This can be accomplished by adding a "Link" header field ([RFC5988]) to a subsequent request to the origin server, detailing the URI of the secondary resource and the failure reason.

The following link extension relations are defined:

[[purl: purl.org seems to have turned read-only; we may need a different way to mint identifiers]]

[[erwip: This is a rough proposal for an error reporting mechanism. Is it good enough? Is it needed at all? Note that Alt-Svc doesn't have anything like this.]]

3.4.1. Server Not Reachable

Used in case the server was not reachable.

Link relation:

http://purl.org/NET/linkrel/not-reachable

3.4.2. Resource Not Found

Used in case the server responded, but the object could not be obtained.

Link relation:

http://purl.org/NET/linkrel/resource-not-found

3.4.3. Payload Unusable

Used in case the payload could be obtained, but wasn't usable (for
instance, because integrity checks failed).

Link relation:

http://purl.org/NET/linkrel/payload-unusable

3.4.4. TLS Handshake Failure

Used in case of a TLS handshare failure ([RFC5246]).

Link relation:

http://purl.org/NET/linkrel/tls-handshake-failure

3.5. Examples

3.5.1. Basic Example

Client request of primary resource at https://www.example.com/test:

```
GET /test HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, out-of-band
```

Response:

```
HTTP/1.1 200 OK
Date: Thu, 14 May 2015 18:52:00 GMT
Content-Type: text/plain
Cache-Control: max-age=10, public
Content-Encoding: out-of-band
Content-Length: 133
Vary: Accept-Encoding

{
  "sr": [
    "http://example.net/bae27c36-fa6a-11e4-ae5d-00059a3c7a00",
    "/c/bae27c36-fa6a-11e4-ae5d-00059a3c7a00"
  ]
}
```

(note that the Content-Type header field describes the media type of
the secondary's resource representation, and the origin server
supplied a fallback URI)

Client request for secondary resource:

```
GET /bae27c36-fa6a-11e4-ae5d-00059a3c7a00 HTTP/1.1
Host: example.net
Origin: https://www.example.com
```

Response:

```
HTTP/1.1 200 OK
Date: Thu, 14 May 2015 18:52:10 GMT
Cache-Control: private
Content-Length: 15

Hello, world.
```

(Note no Content-Type header field is present here because the secondary server truly does not know the media type of the payload)

Final message after recombining header fields:

```
HTTP/1.1 200 OK
Date: Thu, 14 May 2015 18:52:00 GMT
Content-Length: 15
Cache-Control: max-age=10, public
Content-Type: text/plain

Hello, world.
```

3.5.2. Example for an attempt to use 'out-of-band' cross-origin

Section 3.3 requires the client to include an "Origin" header field in the request to a secondary server. The example below shows how the server for the secondary resource would respond to a request which contains an "Origin" header field identifying an unauthorized origin.

Continuing with the example from Section 3.5.1, and a secondary server that is configured to allow only access for requests initiated by "https://www.example.org":

Client request for secondary resource:

```
GET /bae27c36-fa6a-11e4-ae5d-00059a3c7a00 HTTP/1.1
Host: example.net
Origin: https://www.example.com
```

Response:

```
HTTP/1.1 403 Forbidden
Date: Thu, 14 May 2015 18:52:10 GMT
```

Note that a request missing the "Origin" header field would be
treated the same way.

[[anchor6: Any reason why to *mandate* a specific 4xx code?]]

3.5.3. Example involving an encrypted resource

Given the example HTTP message from Section 5.4 of [ENCRYPTENC], a
primary resource could use the 'out-of-band' coding to specify just
the location of the secondary resource plus the contents of the
"Crypto-Key" header field needed to decrypt the payload:

Response:

```
HTTP/1.1 200 OK
Date: Thu, 14 May 2015 18:52:00 GMT
Content-Encoding: aesgcm, out-of-band
Content-Type: text/plain
Encryption: keyid="a1"; salt="vr0o6Uq3w_KDWeatc27mUg"
Crypto-Key: keyid="a1"; aesgcm="csPJEXBYA5U-Tal9EdJi-w"
Content-Length: 85
Vary: Accept-Encoding

{
  "sr": [
    "http://example.net/bae27c36-fa6a-11e4-ae5d-00059a3c7a00"
  ]
}
```

(note that the Content-Type header field describes the media type of
the secondary's resource representation)

Response for secondary resource:

```
HTTP/1.1 200 OK
Date: Thu, 14 May 2015 18:52:10 GMT
Content-Length: ...

VDeU0XxaJkOJDAxPl7h9JD5V8N43RorP7PfpPdZZQuwF
(payload body shown in base64 here)
```

   Final message undoing all content codings:

      HTTP/1.1 200 OK
      Date: Thu, 14 May 2015 18:52:00 GMT
      Content-Length: 15
      Content-Type: text/plain

      I am the walrus

       Note: in this case, the ability to undo the 'aesgcm' is needed to
       process the response. If 'aesgcm' wasn't listed as acceptable
       content coding in the request, the origin server wouldn't be able
       to use the 'out-of-band' mechanism.

3.5.4. Example For Problem Reporting

   Client requests primary resource as in Section 3.5.1, but the attempt
   to access the secondary resource fails.

   Response:

      HTTP/1.1 404 Not Found
      Date: Thu, 08 September 2015 16:49:00 GMT
      Content-Type: text/plain
      Content-Length: 20

      Resource Not Found

   Client retries with the origin server and includes Link header field
   reporting the problem:

      GET /test HTTP/1.1
      Host: www.example.com
      Accept-Encoding: gzip, out-of-band
      Link: <http://example.net/bae27c36-fa6a-11e4-ae5d-00059a3c7a00>;
            rel="http://purl.org/NET/linkrel/resource-not-found"

3.5.5. Relation to Content Negotiation

   Use of the 'out-of-band' encoding is a case of "proactive content
   negotiation", as defined in Section 3.4 of [RFC7231].

   This however does not rule out combining it with other content
   codings. As an example, the possible iterations with the 'gzip'
   content coding ([RFC7230], Section 4.2.3) are described below:

   Case 1: Primary resource does not support 'gzip' encoding

   In this case, the response for the primary resource will never
   include 'gzip' in the Content-Encoding header field. The secondary
   resource however might support it, in which case the client could
   negotiate compression by including "Accept-Encoding: gzip" in the
   request to the secondary resource.

   Case 2: Primary resource does support 'gzip' encoding

   Here, the origin server would actually use two different secondary
   resources, one of them being gzip-compressed. For instance -- going
   back to the first example in Section 3.5.1 -- it might reply with:

```
     HTTP/1.1 200 OK
     Date: Thu, 14 May 2015 18:52:00 GMT
     Content-Type: text/plain
     Cache-Control: max-age=10, public
     Content-Encoding: gzip, out-of-band
     Content-Length: 133
     Vary: Accept-Encoding

     {
       "sr": [
         "http://example.net/bae27c36-fa6a-11e4-ae5d-00059a3c7a01",
         "/c/bae27c36-fa6a-11e4-ae5d-00059a3c7a01"
       ]
     }
```

   which would mean that the payload for the secondary resource already
   is gzip-compressed.

      Note: The origin server could also apply gzip compression to the
      out-of-band payload, in which case the Content-Encoding field
      value would become: "gzip, out-of-band, gzip".

4. Content Codings and Range Requests

   The combination of content codings ([RFC7231], Section 3.1.2 with
   range requests ([RFC7233]) can lead to surprising results, as
   applying the range request happens after applying content codings.

   Thus, for a request for the bytes starting at position 100000 of a
   video:

```
     GET /test.mp4 HTTP/1.1
     Host: www.example.com
     Range: bytes=100000-
     Accept-Encoding: identity
```

...a successful response would use status code 206 (Partial Content)
and have a payload containing the octets starting at position 100000.

```
HTTP/1.1 206 Partial Content
Date: Thu, 08 September 2015 16:49:00 GMT
Content-Type: video/mp4
Content-Length: 134567
Content-Range: bytes 100000-234566/234567

(binary data)
```

However, if the request would have allowed the use of 'out-of-band'
coding:

```
GET /test.mp4 HTTP/1.1
Host: www.example.com
Range: bytes=100000-
Accept-Encoding: out-of-band
```

...a server might return an empty payload (if the out-of-band coded
response body would be shorter than 100000 bytes, as would be usually
the case).

Thus, in order to avoid unnecessary network traffic, servers SHOULD
NOT apply range request processing to responses using ouf-of-band
content coding (or, in other words: ignore "Range" request header
fields in this case).

5. Feature Discovery

New content codings can be deployed easily, as the client can use the
"Accept-Encoding" header field (Section 5.3.4 of [RFC7231]) to signal
which content codings are supported.

6. Security Considerations

6.1. Content Modifications

This specification does not define means to verify that the payload
obtained from the secondary resource really is what the origin server
expects it to be. Content signatures can address this concern (see
[CONTENTSIG] and [MICE]).

6.2. Content Stealing

The 'out-of-band' content coding could be used to circumvent the
same-origin policy ([RFC6454], Section 3) of user agents: an

attacking site which knows the URI of a secondary resource would use
the 'out-of-band' coding to trick the user agent to read the contents
of the secondary resource, which then, due to the security properties
of this coding, would be handled as if it originated from the
origin's resource.

This scenario is addressed by the client requirement to include the
"Origin" request header field and the server requirement to verify
that the request was initiated by an authorized origin.

   Note: similarities with the "Cross-Origin Resource Sharing"
   protocol ([CORS]) are intentional.

Requiring the secondary resource's payload to be encrypted
([ENCRYPTENC]) is an additional mitigation.

6.3. Use in Requests

In general, content codings can be used in both requests and
responses. This particular content coding has been designed for
responses. When supported in requests, it creates a new attack
vector where the receiving server can be tricked into including
content that the client might not have access to otherwise (such as
HTTP resources behind a firewall).

7. IANA Considerations

The IANA "HTTP Content Coding Registry", located at
<http://www.iana.org/assignments/http-parameters>, needs to be
updated with the registration below:

Name: out-of-band

Description: Payload needs to be retrieved from a secondary resource

Reference: Section 3 of this document

8. References

8.1. 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>.

   [RFC5988]   Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/
                    RFC5988, October 2010,
                    <http://www.rfc-editor.org/info/rfc5988>.

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

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

   [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>.

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

   [RFC7235]   Fielding, R., Ed. and J. Reschke, Ed., "Hypertext
                    Transfer Protocol (HTTP/1.1): Authentication",
                    RFC 7235, DOI 10.17487/RFC7235, June 2014,
                    <http://www.rfc-editor.org/info/rfc7235>.

8.2. Informative References

   [CONTENTSIG]   Thomson, M., "Content-Signature Header Field for HTTP",
                    draft-thomson-http-content-signature-00 (work in
                    progress), July 2015.

   [CORS]   van Kesteren, A., "Cross-Origin Resource Sharing", W3C
                    Recommendation REC-cors-20140116, January 2014,
                    <http://www.w3.org/TR/2014/REC-cors-20140116/>.

                    Latest version available at
                    <http://www.w3.org/TR/cors/>.

   [ENCRYPTENC]   Thomson, M., "Encrypted Content-Encoding for HTTP",
                    draft-ietf-httpbis-encryption-encoding-02 (work in
                    progress), June 2016.

   [MICE]   Thomson, M., "Merkle Integrity Content Encoding",

                   draft-thomson-http-mice-01 (work in progress),
                   June 2016.

   [RFC2017]   Freed, N. and K. Moore, "Definition of the URL MIME
                   External-Body Access-Type", RFC 2017, DOI 10.17487/
                   RFC2017, October 1996,
                   <http://www.rfc-editor.org/info/rfc2017>.

   [RFC4483]   Burger, E., "A Mechanism for Content Indirection in
                   Session Initiation Protocol (SIP) Messages", RFC 4483,
                   DOI 10.17487/RFC4483, May 2006,
                   <http://www.rfc-editor.org/info/rfc4483>.

   [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>.

   [RFC6454]   Barth, A., "The Web Origin Concept", RFC 6454,
                   DOI 10.17487/RFC6454, December 2011,
                   <http://www.rfc-editor.org/info/rfc6454>.

   [RFC7232]   Fielding, R., Ed. and J. Reschke, Ed., "Hypertext
                   Transfer Protocol (HTTP/1.1): Conditional Requests",
                   RFC 7232, DOI 10.17487/RFC7232, June 2014,
                   <http://www.rfc-editor.org/info/rfc7232>.

   [RFC7233]   Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
                   "Hypertext Transfer Protocol (HTTP/1.1): Range
                   Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014,
                   <http://www.rfc-editor.org/info/rfc7233>.

   [SCD]   Thomson, M., Eriksson, G., and C. Holmberg, "An
                   Architecture for Secure Content Delegation using HTTP",
                   draft-thomson-http-scd-01 (work in progress),
                   June 2016.

URIs

   [1] <mailto:ietf-http-wg@w3.org>

   [2] <mailto:ietf-http-wg-request@w3.org?subject=subscribe>

Appendix A. Alternatives, or: why not a new Status Code?

   A plausible alternative approach would be to implement this
   functionality one level up, using a new redirect status code (Section
   6.4 of [RFC7231]). However, this would have several drawbacks:

   o Servers will need to know whether a client understands the new
      status code; thus some additional signal to opt into this protocol
      would always be needed.

   o In redirect messages, representation metadata (Section 3.1 of
      [RFC7231]), namely "Content-Type", applies to the response
      message, not the redirected-to resource.

   o The origin-preserving nature of using a content coding would be
      lost.

   Another alternative would be to implement the indirection on the
   level of the media type using something similar to the type "message/
   external-body", defined in [RFC2017] and refined for use in the
   Session Initiation Protocol (SIP) in [RFC4483]. This approach though
   would share most of the drawbacks of the status code approach
   mentioned above.

Appendix B. Open Issues

B.1. Accessing the Secondary Resource Too Early

   One use-case for this protocol is to enable a system of "blind
   caches", which would serve the secondary resources. These caches
   might only be populated on demand, thus it could happen that whatever
   mechanism is used to populate the cache hasn't finished when the
   client hits it (maybe due to race conditions, or because the cache is
   behind a middlebox which doesn't allow the origin server to push
   content to it).

   In this particular case, it can be useful if the client was able to
   "piggyback" the URI of the fallback for the primary resource, giving
   the secondary server a means by which it could obtain the payload
   itself. This information could be provided in yet another Link
   header field:

```
     GET /bae27c36-fa6a-11e4-ae5d-00059a3c7a00 HTTP/1.1
     Host: example.net
     Link: <http://example.com/c/bae27c36-fa6a-11e4-ae5d-00059a3c7a00>;
           rel="http://purl.org/NET/linkrel/fallback-resource"
```

   (continuing the example from Section 3.5.1)

B.2. Resource maps

   When 'out-of-band' coding is used as part of a caching solution, the
   additional round trips to the origin server can be a significant

performance problem; in particular, when many small resources need to be loaded (such as scripts, images, or video fragments). In cases like these, it could be useful for the origin server to provide a "resource map", allowing to skip the round trips to the origin server for these mapped resources. Plausible ways to transmit the resource map could be:

o as extension in the 'out-of-band' coding JSON payload, or

o as separate resource identified by a "Link" response header field.

This specification does not define a format, nor a mechanism to transport the map, but it's a given that some specification using 'out-of-band' coding will do.

B.3. Fragmenting

It might be interesting to divide the original resource's payload into fragments, each of which being mapped to a distinct secondary resource. This would allow to not store the full payload of a resource in a single cache, thus

o distribute load,

o caching different parts of the resource with different characteristics (such as only distribute the first minutes of a long video), or

o fetching specific parts of a resource (similar to byte range requests), or

o hiding information from the secondary server.

Another benefit might be that it would allow the origin server to only serve the first part of a resource itself (reducing time to play of a media resource), while delegating the remainder to a cache (however, this might require further adjustments of the 'out-of-band' payload format).

B.4. Relation to Content Encryption

Right now this specification is orthogonal to [ENCRYPTENC]/[MICE]; that is, it could be used for public content such as software downloads. However, the lack of mandatory encryption affects the security considerations (which currently try to rule attack vectors caused by ambient authority ([RFC6265], Section 8.2). We need to decide whether we need this level of independence.

B.5. **Reporting**

   This specification already defines hooks through which a client can
   report failures when accessing secondary resources (see Section 3.4).

   However, it would be useful if there were also ways to report on
   statistics such as:

   o Success (Cache Hit) rates, and

   o Bandwidth to secondary servers.

   This could be implemented using a new service endpoint and a (JSON?)
   payload format.

   Similarly, a reporting facility for use by the secondary servers
   could be useful.

Appendix C.   **Change Log (to be removed by RFC Editor before publication)**

C.1. **Changes since draft-reschke-http-oob-encoding-00**

   Mention media type approach.

   Explain that clients can always fall back not to use oob when the
   secondary resource isn't available.

   Add Vary response header field to examples and mention that it'll
   usually be needed
   (<https://github.com/reschke/oobencoding/issues/6>).

   Experimentally add problem reporting using piggy-backed Link header
   fields (<https://github.com/reschke/oobencoding/issues/7>).

C.2. **Changes since draft-reschke-http-oob-encoding-01**

   Updated ENCRYPTENC reference.

C.3. **Changes since draft-reschke-http-oob-encoding-02**

   Add MICE reference.

   Remove the ability of the secondary resource to contain anything but
   the payload (<https://github.com/reschke/oobencoding/issues/11>).

   Changed JSON payload to be an object containing an array of URIs plus
   additional members. Specify "fallback" as one of these additional
   members, and update Appendix B.1 accordingly).

Discuss extensibility a bit.

C.4.   **Changes since draft-reschke-http-oob-encoding-03**

Mention "Content Stealing" thread.

Mention padding.

C.5.   **Changes since draft-reschke-http-oob-encoding-04**

Reduce information leakage by disallowing ambient authority
information being sent to the secondary resource. Require "Origin"
to be included in request to secondary resource, and require
secondary server to check it.

Mention "Origin" + server check on secondary resource as defense to
content stealing.

Update ENCRYPTENC reference, add SCD reference.

Mention fragmentation feature.

Discuss relation with range requests.

C.6.   **Changes since draft-reschke-http-oob-encoding-05**

Remove redundant Cache-Control: private from one example response
(the response payload is encrypted anyway).

Mention looping.

Remove 'metadata' payload element.

Align with changes in ENCRYPTENC spec.

Fix incorrect statement about what kind of cookies/credentials can be
used in the request to the secondary resource.

Rename "URIs" to "sr" ("secondary resources") and treat the fallback
URI like a regular secondary resource.

Mention reporting protocol ideas.

C.7.   **Changes since draft-reschke-http-oob-encoding-06**

Changed the link relation name to the fallback resource from
"primary" to "fallback". Added link relation for reporting TLS
handshake failures.

Added an example about the interaction with 'gzip' coding.

Update ENCRYPTENC, MICE, and SCD references.

Appendix D. Acknowledgements

Thanks to Christer Holmberg, Daniel Lindstrom, Erik Nygren, Goran Eriksson, John Mattsson, Kevin Smith, Magnus Westerlund, Mark Nottingham, Martin Thomson, and Roland Zink for feedback on this document.

Authors' Addresses

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

Salvatore Loreto
Ericsson
Torshamnsgatan 21
Stochholm 16483
Sweden

EMail: salvatore.loreto@ericsson.com