OHAI Working Group
Internet-Draft
Intended status: Standards Track
Expires: 3 January 2026

T. Pauly
Apple
M. Thomson
Mozilla

2 July 2025

Chunked Oblivious HTTP Messages
draft-ietf-ohai-chunked-ohttp-05

Abstract

This document defines a variant of the Oblivious HTTP message format that allows chunks of requests and responses to be encrypted and decrypted before the entire request or response is processed. This allows incremental processing of Oblivious HTTP messages, which is particularly useful for handling large messages or systems that process messages slowly.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-wg-ohai.github.io/draft-ohai-chunked-ohttp/draft-ietf-ohai-chunked-ohttp.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-ohai-chunked-ohttp/.

Discussion of this document takes place on the OHAI Working Group mailing list (mailto:ohai@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/ohai/. Subscribe at https://www.ietf.org/mailman/listinfo/ohai/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-ohai/draft-ohai-chunked-ohttp.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Oblivious HTTP [OHTTP] defines a system for sending HTTP requests and responses as encrypted messages. Clients send requests via a relay to a gateway, which is able to decrypt and forward the request to a target server. Responses are encrypted with an ephemeral symmetric key by the gateway and sent back to the client via the relay. The messages are protected with Hybrid Public Key Encryption (HPKE; [HPKE]), and are intended to prevent the gateway from linking any two independent requests to the same client.

The definition of Oblivious HTTP in [OHTTP] encrypts messages such that entire request and response bodies need to be received before any of the content can be decrypted. This is well-suited for many of the use cases of Oblivious HTTP, such as DNS queries or metrics reporting.

However, some applications of Oblivious HTTP can benefit from being able to encrypt and decrypt parts of the messages in chunks. If a request or response can be processed by a receiver in separate parts, and is particularly large or will be generated slowly, then sending a series of encrypted chunks can improve the performance of applications.

Incremental delivery of responses allows an Oblivious Gateway Resource to provide Informational (1xx) responses (Section 15.2 of [HTTP]).

This document defines an optional message format for Oblivious HTTP that supports the progressive creation and processing of both requests and responses. New media types are defined for this purpose.

1.1. Applicability

Like the non-chunked variant, chunked Oblivious HTTP has limited applicability as described in Section 2.1 of [OHTTP], and requires the use of a willing Oblivious Relay Resource and Oblivious Gateway Resource.

Chunked Oblivious HTTP is intended to be used in cases for where the privacy properties of Oblivious HTTP are needed — specifically, removing linkage at the transport layer between separate HTTP requests — but incremental processing is also needed for performance or functionality.

One specific functional capability that requires chunked Oblivious HTTP is support for Informational (1xx) responses (Section 15.2 of [HTTP]).

In order to be useful, the content of chunked Oblivious HTTP needs to be possible to process incrementally. Since incremental processing means that the message might end up being truncated, for example in the case of an error on the underlying transport, applications also need to be prepared to safely handle incomplete messages (see Section 7 for more discussion). Choices about how the inner content is structured can be made independently of this chunked format; that is, Binary HTTP chunks do need not to align with those of OHTTP.

Applications that use the Indeterminate format of Binary HTTP (Section 3.2 of [BHTTP]) are well-suited to using chunked Oblivious HTTP as it enables incremental construction of messages. That only applies to construction; how a message can be processed after decryption depends on how the format is processed. Binary HTTP messages in any format (either Known- or Indeterminate-Length) can be incrementally processed.

Chunked Oblivious HTTP is not intended to be used for long-lived sessions between clients and servers that might build up state, or as a replacement for a proxied TLS session.

2. Conventions and Definitions

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

Notational conventions from [OHTTP] are used in this document.

3. Chunked Requests and Responses

Chunked Oblivious HTTP defines different media than the non-chunked variant. These media types are "message/ohttp-chunked-req" (defined in Section 8.1) and "message/ohttp-chunked-res" (defined in Section 8.2). If a request uses the media type "message/ohttp-chunked-req", a successful corresponding response MUST use the media type "message/ohttp-chunked-res".

Use cases that require the use of Chunked OHTTP SHOULD only use the chunked media types for their requests, to indicate that Chunked OHTTP is required. If the gateway unexpectedly does not support Chunked OHTTP, then the request will fail as if OHTTP as a whole were

not supported. If clients retry requests with the non-chunked media type, a gateway could partition client anonymity sets by rejecting some requests and accepting others.

Chunked OHTTP requests and responses SHOULD include the Incremental header field [INCREMENTAL] in order to signal to intermediaries (such as the relay) that the content of the messages are intended to be delivered incrementally. Without this signal, intermediaries might buffer request or response body until complete, removing the benefits of using Chunked OHTTP.

Chunked OHTTP messages generally will not include a Content-Length header field, since the complete length of all chunks might not be known ahead of time.

For example, a Chunked OHTTP request could look like the following:

```
POST /request.example.net/proxy HTTP/1.1
Host: proxy.example.org
Content-Type: message/ohttp-chunked-req
Incremental: ?1
Transfer-Encoding: chunked

<content is an Encapsulated Request>
```

Implementations MUST support receiving chunks that contain 2^14 (16384) octets of data prior to encapsulation. Senders of chunks SHOULD limit their chunks to this size, unless they are aware of support for larger sizes by the receiving party.

4. Request Format

Chunked OHTTP requests start with the same header as used for the non-chunked variant, which consists of a key ID, algorithm IDs, and the KEM shared secret. This header is followed by chunks of data protected with HPKE, each of which is preceded by a variable-length integer (as defined in Section 16 of [QUIC]) that indicates the length of the chunk. The final chunk is preceded by a length field with the value 0, which means the chunk extends to the end of the outer stream.

```
Chunked Encapsulated Request {
  Chunked Request Header (56 + 8 * Nenc),
  Chunked Request Chunks (..),
}

Chunked Request Header {
  Key Identifier (8),
  HPKE KEM ID (16),
  HPKE KDF ID (16),
  HPKE AEAD ID (16),
  Encapsulated KEM Shared Secret (8 * Nenc),
}

Chunked Request Chunks {
  Non-Final Request Chunk (..),
  Final Request Chunk Indicator (i) = 0,
  HPKE-Protected Final Chunk (..),
}

Non-Final Request Chunk {
  Length (i) = 1..,
  HPKE-Protected Chunk (..),
}
```

Figure 1: Chunked Encapsulated Request Format

The content of the HPKE-protected chunks is defined in Section 6.1.

5. Response Format

Chunked OHTTP responses start with a nonce, followed by chunks of data protected with an AEAD. Each chunk is preceded by a variable-length integer that indicates the length of the chunk. The final chunk is preceded by a length field with the value 0, which means the chunk extends to the end of the outer stream.

```
Chunked Encapsulated Response {
  Response Nonce (Nk),
  Chunked Response Chunks (..),
}

Chunked Response Chunks {
  Non-Final Response Chunk (..),
  Final Response Chunk Indicator (i) = 0,
  AEAD-Protected Final Response Chunk (..),
}

Non-Final Response Chunk {
  Length (i) = 1..,
  AEAD-Protected Chunk (..),
}
```

Figure 2: Chunked Encapsulated Response Format

6. Encapsulation of Chunks

The encapsulation of chunked Oblivious HTTP requests and responses uses the same approach as the non-chunked variant, with the difference that the body of requests and responses are sealed and opened in chunks, instead of as a whole.

The AEAD that protects both requests and responses protects individual chunks from modification or truncation. Additionally, chunk authentication protects two other pieces of information:

1. the order of the chunks (the sequence number of each chunk), which is included in the nonce of each chunk.

2. which chunk is the final chunk, which is indicated by a sentinel in the Additional Authenticated Data (AAD) of the final chunk.

The format of the outer packaging that carries the chunks (the length prefix for each chunk specifically) is not explicitly authenticated. This allows the chunks to be transported by alternative means, and still be valid as long as the order and finality are preserved. In particular, the variable-length encoding used for lengths allows for different expressions of the same value, where the choice between equivalent encodings is not authenticated.

6.1. Request Encapsulation

For requests, the setup of the HPKE context and the encrypted request header is the same as the non-chunked variant. This is the Chunked Request Header defined in Section 4.

```
hdr = concat(encode(1, key_id),
             encode(2, kem_id),
             encode(2, kdf_id),
             encode(2, aead_id))
info = concat(encode_str("message/bhttp chunked request"),
              encode(1, 0),
              hdr)
enc, sctxt = SetupBaseS(pkR, info)
enc_request_hdr = concat(hdr, enc)
```

Each chunk is sealed using the HPKE context. For non-final chunks, the AAD is empty.

```
sealed_chunk = sctxt.Seal("", chunk)
sealed_chunk_len = varint_encode(len(sealed_chunk))
non_final_chunk = concat(sealed_chunk_len, sealed_chunk)
```

The final chunk in a request uses an AAD of the string "final" and is prefixed with a zero length.

```
sealed_final_chunk = sctxt.Seal("final", chunk)
final_chunk = concat(varint_encode(0), sealed_final_chunk)
```

HPKE already maintains a sequence number for sealing operations as part of the context, so the order of chunks is protected. HPKE will produce an error if the sequence number overflows, which puts a limit on the number of chunks that can be sent in a request.

6.2. Response Encapsulation

For responses, the first piece of data sent back is the response nonce, as in the non-chunked variant. As in the non-chunked variant, the length of the nonce is max(Nn, Nk), where Nn and Nk are the length of the AEAD nonce and key.

```
entropy_len = max(Nn, Nk)
response_nonce = random(entropy_len)
```

Each chunk is sealed using the same AEAD key and AEAD nonce that are derived for the non-chunked variant, which are calculated as follows:

```
secret = context.Export("message/bhttp chunked response", entropy_len)
salt = concat(enc, response_nonce)
prk = Extract(salt, secret)
aead_key = Expand(prk, "key", Nk)
aead_nonce = Expand(prk, "nonce", Nn)
```

The sender also maintains a counter of chunks, which is set to 0 for the first chunk an incremented by 1 after encoding each chunk.

```
counter = 0
```

The AEAD nonce is XORed with the counter for encrypting (and decrypting) each chunk. For non-final chunks, the AAD is empty.

```
chunk_nonce = aead_nonce XOR encode(Nn, counter)
sealed_chunk = Seal(aead_key, chunk_nonce, "", chunk)
sealed_chunk_len = varint_encode(len(sealed_chunk))
non_final_chunk = concat(sealed_chunk_len, sealed_chunk)
counter++
```

The final chunk in a response uses an AAD of the string "final" and is prefixed with a zero length.

```
chunk_nonce = aead_nonce XOR encode(Nn, counter)
sealed_final_chunk = Seal(aead_key, chunk_nonce, "final", chunk)
final_chunk = concat(varint_encode(0), sealed_final_chunk)
```

If the counter reached the maximum value that can be held in an integer with Nn bits (that maximum being 2^Nn), where Nn is the length of the AEAD nonce, the chunk_nonce would wrap and be reused. Therefore, the response MUST NOT use 2^Nn or more chunks.

7. Security Considerations

In general, Chunked OHTTP inherits the same security considerations as Oblivious HTTP [OHTTP]. Note specifically that while Chunked OHTTP allows for incremental delivery and processing of messages, it does not add forward secrecy between chunks. As with the non-chunked variant, forward secrecy is only provided when changing the key configuration. This is particularly important when chunking is used to enable interactivity.

The use of Chunked OHTTP can be considered part of the configuration a client knows about for a particular gateway. As such, the use of Chunked OHTTP falls under the same consistency privacy considerations as the rest of the configuration (see Section 7 of [OHTTP]). Specifically, clients SHOULD NOT fall back from Chunked OHTTP to the non-chunked variant if they are configured to used chunking. Falling back would allow clients to have inconsistent behavior that could be used to partition client anonymity sets.

7.1. Message Truncation

The primary advantage of a chunked encoding is that chunked requests or responses can be generated or processed incrementally. However, for a recipient in particular, processing an incomplete message can have security consequences.

The potential for message truncation is not a new concern for HTTP. All versions of HTTP provide incremental delivery of messages. For this use of Oblivious HTTP, incremental processing that might result in side-effects demands particular attention as Oblivious HTTP does not provide strong protection against replay attacks; see Section 6.5 of [OHTTP]. Truncation might be the result of interference at the network layer, or by a malicious Oblivious Relay Resource.

Endpoints that receive chunked messages can perform early processing if the risks are understood and accepted. Conversely, endpoints that depend on having a complete message MUST ensure that they do not consider a message complete until having received a chunk with a 0-valued length prefix, which was successfully decrypted using the expected sentinel value, "final", in the AAD.

7.2. Interactivity and Privacy

Without chunking, Oblivious HTTP involves a single request and response, with no further interactivity. Using a chunked variant at both Client and Oblivious Gateway Resource creates the possibility that an exchange could lead to multiple rounds of interaction. Information from early chunks from a peer could influence how an endpoint constructs later chunks of their message. However, the use of Chunked OHTTP does not necessarily mean that exchanges will involve interactivity.

Interactivity for Chunked OHTTP can be defined as any case in which the response can influence the timing or content of the request. To help explain this distinction, the following scenarios can be used to understand different modalities for requests and responses:

* The request is sent as a single chunk, and the response is sent as a single chunk. This is a non-interactive case that is identical to the non-chunked variant.

* The request is sent as a single chunk, and the response is sent in multiple chunks. This is a non-interactive case, because there is no possibility that the client can influence its request based on the response content.

* The request is sent in multiple chunks, but either all chunks are sent before a response chunk is received, or the sending of the chunks is not influenced by the response chunks. This is a non-interactive case, since again the client's request is not influenced by any response content.

* The request is sent in multiple chunks, at least one of which specifically is sent after receiving -- and possibly processing -- a response chunk (or the complete response), where the response influences the timing and/or content of the request chunk. This is an interactive case.

In the interactive case, the Oblivious Gateway Resource can observe the round trip time to the Client, which can change the privacy assumptions of the system.

Any interactivity also allows a network adversary (including the Oblivious Relay Resource) to measure the round-trip delay from themselves to the Client.

Client implementations therefore need to be aware of the possibility that interactively processing chunks might reveal round-trip time information that would be kept private in a non-interactive exchange.

For cases when interactivity introduces unacceptable risks, the client can ensure that it never has an interactive exchange, either by not sending its request in multiple chunks, or by ensuring that the sending of request chunks cannot be influenced by the response.

Interactivity that is deliberate might be acceptable. For instance, the 100-continue feature in HTTP, which has the client withhold the body of a request until it receives a 100 Informational response, is not possible without an interactive exchange. This highlights the risks involved in the use of this chunked encoding to adapt an existing HTTP-based interaction to use Oblivious HTTP as such an adaptation might not achieve expected privacy outcomes.

Interactivity does not inherently reduce replay risk unless the server explicitly verifies that a client is live (such as by having the client echo content from the response in its request). A request that is generated interactively can be replayed by a malicious relay.

8. IANA Considerations

This document updates the "Media Types" registry at https://iana.org/assignments/media-types (https://iana.org/assignments/media-types) to add the media types "message/ohttp-chunked-req" (Section 8.1), and "message/ohttp-chunked-res" (Section 8.2), following the procedures of [RFC6838].

8.1. message/ohttp-chunked-req Media Type

The "message/ohttp-chunked-req" identifies an encrypted binary HTTP request that is transmitted or processed in chunks. This is a binary format that is defined in Section 4.

Type name: message
Subtype name: ohttp-chunked-req
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: "binary"
Security considerations: see Section 7
Interoperability considerations: N/A
Published specification: this specification
Applications that use this media type: Oblivious HTTP and applications that use Oblivious HTTP use this media type to identify encapsulated binary HTTP requests that are incrementally generated or processed.
Fragment identifier considerations: N/A
Additional information: Magic number(s): N/A
Deprecated alias names for this type: N/A
File extension(s): N/A
Macintosh file type code(s): N/A
Person and email address to contact for further information: see Aut hors' Addresses section
Intended usage: COMMON
Restrictions on usage: N/A
Author: see Authors' Addresses section
Change controller: IETF

8.2. message/ohttp-chunked-res Media Type

The "message/ohttp-chunked-res" identifies an encrypted binary HTTP response that is transmitted or processed in chunks. This is a binary format that is defined in Section 5.

Type name: message
Subtype name: ohttp-chunked-res
Required parameters: N/A
Optional parameters: N/A

Encoding considerations: "binary"
Security considerations: see Section 7
Interoperability considerations: N/A
Published specification: this specification
Applications that use this media type: Oblivious HTTP and applications that use Oblivious HTTP use this media type to identify encapsulated binary HTTP responses that are incrementally generated or processed.
Fragment identifier considerations: N/A
Additional information: Magic number(s): N/A
Deprecated alias names for this type: N/A
File extension(s): N/A
Macintosh file type code(s): N/A
Person and email address to contact for further information: see Authors' Addresses section
Intended usage: COMMON
Restrictions on usage: N/A
Author: see Authors' Addresses section
Change controller: IETF

9. References

9.1. Normative References

[BHTTP] Thomson, M. and C. A. Wood, "Binary Representation of HTTP Messages", RFC 9292, DOI 10.17487/RFC9292, August 2022, <https://www.rfc-editor.org/rfc/rfc9292>.

[HPKE] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[INCREMENTAL] Oku, K., Pauly, T., and M. Thomson, "Incremental HTTP Messages", Work in Progress, Internet-Draft, draft-kazuho-httpbis-incremental-http-00, 15 October 2024, <https://datatracker.ietf.org/doc/html/draft-kazuho-httpbis-incremental-http-00>.

[OHTTP] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, DOI 10.17487/RFC9458, January 2024, <https://www.rfc-editor.org/rfc/rfc9458>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

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

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

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

9.2. Informative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

Appendix A. Example

A single request and response exchange is shown here. This follows the same basic setup as the example in Appendix A of [OHTTP].

The Oblivious Gateway Resource key pair is generated with a X25519 secret key of:

1c190d72acdbe4dbc69e680503bb781a932c70a12c8f3754434c67d8640d8698

The corresponding key configuration is:

010020668eb21aace159803974a4c67f08b4152d29bed10735fd08f98ccdd6fe
0957080008000100010003

This key configuration is somehow obtained by the Client, which constructs a binary HTTP request:

00034745540568747470730b6578616d706c652e636f6d012f

The client constructs an HPKE sending context with a secret key of:

b26d565f3f875ed480d1abced3d665159650c99174fd0b124ac4bda0c64ae324

The corresponding public key is:

8811eb457e100811c40a0aa71340a1b81d804bb986f736f2f566a7199761a032

The context is created with an info parameter of:

6d6573736167652f626874747020636875e6b656420726571756573740001002000010001

This produces an encrypted message, allowing the Client to construct the following Encapsulated Request:

```
01002000010001
8811eb457e100811c40a0aa71340a1b81d804bb986f736f2f566a7199761a032
1c2ad24942d4d692563012f2980c8fef437a336b9b2fc938ef77a5834f
1d2e33d8fd25577afe31bd1c79d094f76b6250ae6549b473ecd950501311
001c6c1395d0ef7c1022297966307b8a7f
```

This message contains a header, the encapsulated secret, and three encrypted chunks. Line breaks are included above to show where these chunks start.

The encrypted chunks are the result of invoking the HPKE ContextS.Seal() function three times: the first with 12 bytes of the request, the second with the remaining 13 bytes, and the last containing no data. This final chunk is marked by a zero length in the encoding and an AAD of "final" to protect against undetected message truncation. Each chunk is expanded by 16 bytes for AEAD protection.

> | A BSD-like read() interface that returns 0 when it reaches the
> | end of a stream naturally leads to a zero-length chunk like
> | this if the data returned is protected immediately.

After sending this to the Oblivious Relay Resource, the Oblivious Gateway Resource decrypts and processes this message. The Target Resource produces a response like:

0140c8

The response is protected by exporting a secret from the HPKE context, using input keying material of:

1d4484834ae36102a6ac42a5523454d9

The salt is:

8811eb457e100811c40a0aa71340a1b81d804bb986f736f2f566a7199761a032bcce7f4cb921309ba5d62edf1769ef09

From these, HKDF-SHA256 produces a pseudorandom key of:

3967884b5f7b4bce4a5320a3e3f79fdc97389f7deba1c1e11c5ea62278187786

The resulting AES-GCM key is:

8209f78f2a1610d80c7125009b00aff0

The 12-byte base nonce is:

fead854635d2d5527d64f546

The AEAD Seal() function is then used to encrypt the response in chunks to produce the Encapsulated Response:

bcce7f4cb921309ba5d62edf1769ef09
1179bf1cc87fa0e2c02de4546945aa3d1e48
12b348b5bd4c594c16b6170b07b475845d1f32
00ed9d8a796617a5b27265f4d73247f639

This example is split onto separate lines to show the nonce and three chunks: the first with one byte of response, the second with the remaining two bytes, and the final with zero bytes of data.

The nonces for processing the chunks are, in order:

fead854635d2d5527d64f546
fead854635d2d5527d64f547
fead854635d2d5527d64f544

Acknowledgments

Thanks to Chris Wood for helping build an initial test implementation and providing reviews. Thanks to Jonathan Hoyland for identifying some of the privacy leaks.

Authors' Addresses

Tommy Pauly
Apple
Email: tpauly@apple.com

Martin Thomson
Mozilla
Email: mt@lowentropy.net