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.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-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/.

Status of This Memo

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

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

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

This Internet-Draft will expire on 18 February 2024.
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). Applications that use the Indeterminate format of Binary HTTP (Section 3.2 of [BHTTP]) are well-suited to using chunked Oblivious HTTP.
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 Request and Response Media Types

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).

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.
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.
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 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".

sealed_final_chunk = sctxt.Seal("final", chunk)
sealed_final_chunk_len = varint_encode(len(sealed_final_chunk))
final_chunk = concat(sealed_final_chunk_len, sealed_final_chunk)

HPKE already maintains a sequence number for sealing operations as part of the context, so the order of chunks is protected.

6.2. Response Encapsulation

For responses, the first piece of data sent back is the response nonce, as in the non-chunked variant.

entropy = max(Nn, Nk)
response_nonce = random(entropy)

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)
response_nonce = random(entropy)
salt = concat(enc, response_nonce)
prk = Extract(salt, secret)
aead_key = Expand(prk, "key", Nk)
aeadNonce = Expand(prk, "nonce", Nn)

The sender also maintains a counter of chunks, which is initialized to 0.

counter = 0
The nonce additionally is XORed with a counter to indicate the order of the chunks. For non-final chunks, the AAD is empty.

```plaintext
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".

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

7. Security Considerations

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.

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 Authors’ 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-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.
9. References

9.1. Normative References


9.2.  Informative References


Acknowledgments

TODO acknowledgements.

Authors’ Addresses

Tommy Pauly
Apple
Email: tpauly@apple.com

Martin Thomson
Mozilla
Email: mt@lowentropy.net