Chunked Oblivious HTTP Messages

draft-ohai-chunked-ohttp

Tommy Pauly & Martin Thomson
OHAI Interim
January 2024
Are we done with OHTTP?
Are we done?

RFC 9458 is published! 🎉

...but there is a functional gap that we can fill in.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Complete messages</th>
<th>Known-length messages</th>
<th>Indeterminate messages</th>
<th>Multiple messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHTTP</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary HTTP</td>
<td></td>
<td>✅</td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>HPKE</td>
<td></td>
<td>✅</td>
<td></td>
<td>✅</td>
</tr>
</tbody>
</table>
Chunked OHTTP

Chunked OHTTP allows encrypting and decrypting requests and responses in separate chunks

  Allows the use of Binary HTTP's "indeterminate" mode

  Takes advantage of HPKE's support for multiple messages

Still is a **single** HTTP request-and-response transaction
What are the use cases?
Example use case #1

Database query that can fetch potentially large responses

Response can be processed incrementally

Response might be slow to return
Example use case #2

Generative content

Content being generated by servers on-the-fly in response to a request

Response might be slow to generate, and sometimes (perhaps not always) large

Clients can start processing or displaying the content incrementally
Example use case #3

Large uploads to a server

Currently, OHTTP has no maximum size, so a server that is willing to accept large requests is forced to buffer unbounded messages before knowing if they can be decrypted, or if they will be served.

Chunking allows the server to validate the HPKE encryption and request headers in the first chunk before accepting the rest of the request.

Chunking also allows endpoints to write data out after decrypting, instead of holding the entire message in memory.
Example use case #4

Informational responses

Support 100-continue from servers so that clients can check that a server is willing to receive a body before they send it

(Warning: server can measure latency to client if the client reaction to a 1xx leaks evident into the request through timing)

Be able to open 1xx responses without waiting for the entire response
To OHTTP, or not to OHTTP
When to use OHTTP

Chunked OHTTP optimizes applications that are already good candidates for OHTTP

This should not replace use cases that would be better served by MASQUE, or other ways to proxy end-to-end TLS streams

There are pros and cons to both approaches, that apply whether or not chunks are used
## Comparing options

<table>
<thead>
<tr>
<th><strong>OHTTP</strong></th>
<th><strong>Proxied TLS (MASQUE)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooperating gateway servers</td>
<td>Unmodified target servers</td>
</tr>
<tr>
<td>Single request/response pair</td>
<td>Bidirectional data stream</td>
</tr>
<tr>
<td>Connection reuse between relay and gateway</td>
<td>Port/IP allocation for each proxied connection</td>
</tr>
<tr>
<td>Per-request decoupling without added latency</td>
<td>Decoupling transactions requires a new TLS handshake</td>
</tr>
<tr>
<td>Gateway cannot measure latency to client (100-continue?)</td>
<td>Target can measure latency to client to infer location</td>
</tr>
<tr>
<td>No replay protection (Date field)</td>
<td>Replay protection</td>
</tr>
<tr>
<td>No PFS until key rotation</td>
<td>PFS</td>
</tr>
</tbody>
</table>
When to use OHTTP

*Don't* use OHTTP for:

- Back-and-forth streaming of data
- Cases where setup latency is relatively unimportant (long-lived streaming, web browsing sessions where you share cookies, etc)
- Cases where replaying the request is dangerous

*Do* use OHTTP for:

- Cases where you want decoupling between HTTP transactions
- Cases where you don't want the server to be able to determine client location based on latency (although *Expect: 100-continue* can undermine this benefit)
- Cases where servers need to scale to handle requests from large numbers of clients
The technical bits
How to chunk OHTTP

Add a varint "length" field before each chunk

Final chunk is indicated by length=0, and extends to the end of the outer stream

Prevents reordering of chunks and truncation by removing chunks

New media types
- `message/ohttp-chunked-req`
- `message/ohttp-chunked-res`
Chunked HPKE Requests

```
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)

for chunk in chunks_except_final:
    sealed_chunk = sctxt.Seal("", chunk)
    sealed_chunk_len = varint_encode(len(sealed_chunk))
    non_final_chunk = concat(sealed_chunk_len, sealed_chunk)

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)
```
Chunked AEAD Responses

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

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)
aead_nonce = Expand(prk, "nonce", Nn)

counter = 0

for chunk in chunks_except_final:
    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++

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)
Interesting questions

Any negotiation or indication of support?

Many uses of OHTTP are some a priori configuration

Implications for discovered support (SVCB)

Should we add prohibitions on sending request chunks after receiving response chunks?
Next steps

Should we adopt?