Stream: Internet Engineering Task Force (IETF)
RFC: 9458
Category: Standards Track
Published: January 2024
ISSN: 2070-1721
Authors: M. Thomson *Mozilla* C. A. Wood *Cloudflare*

RFC 9458
Oblivious HTTP

Abstract

This document describes Oblivious HTTP, a protocol for forwarding encrypted HTTP messages. Oblivious HTTP allows a client to make multiple requests to an origin server without that server being able to link those requests to the client or to identify the requests as having come from the same client, while placing only limited trust in the nodes used to forward the messages.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP requests reveal information about client identities to servers. While the actual content of the request message is under the control of the client, other information that is more difficult to control can still be used to identify the client.

Even where an IP address is not directly associated with an individual, the requests made from it can be correlated over time to assemble a profile of client behavior. In particular, connection reuse improves performance but provides servers with the ability to link requests that share a connection.

In particular, the source IP address of the underlying connection reveals identifying information that the client has only limited control over. While client-configured HTTP proxies can provide a degree of protection against IP address tracking, they present an unfortunate trade-off: if they are used without TLS, the contents of communication are revealed to the proxy; if they are used with TLS, a new connection needs to be used for each request to ensure that the origin server cannot use the connection as a way to correlate requests, incurring significant performance overheads.

To overcome these limitations, this document defines Oblivious HTTP, a protocol for encrypting and sending HTTP messages from a client to a gateway. This uses a trusted relay service in a manner that mitigates the use of metadata such as IP address and connection information for client identification, with reasonable performance characteristics. This document describes:

1. an algorithm for encapsulating binary HTTP messages [BINARY] using Hybrid Public Key Encryption (HPKE) [HPKE] to protect their contents,
2. a method for forwarding Encapsulated Requests between Clients and an Oblivious Gateway Resource through a trusted Oblivious Relay Resource using HTTP, and
3. requirements for how the Oblivious Gateway Resource handles Encapsulated Requests and produces Encapsulated Responses for the Client.

The combination of encapsulation and relaying ensures that Oblivious Gateway Resource never sees the Client's IP address and that the Oblivious Relay Resource never sees plaintext HTTP message content.

Oblivious HTTP allows connection reuse between the Client and Oblivious Relay Resource, as well as between that relay and the Oblivious Gateway Resource, so this scheme represents a performance improvement over using just one request in each connection. With limited trust placed in the Oblivious Relay Resource (see Section 6), Clients are assured that requests are not uniquely attributed to them or linked to other requests.

2. Overview

An Oblivious HTTP Client must initially know the following:

- The identity of an Oblivious Gateway Resource. This might include some information about what Target Resources the Oblivious Gateway Resource supports.
- The details of an HPKE public key for the Oblivious Gateway Resource, including an identifier for that key and the HPKE algorithms that are used with that key.
- The identity of an Oblivious Relay Resource that will accept relay requests carrying an Encapsulated Request as its content and forward the content in these requests to a particular Oblivious Gateway Resource. Oblivious HTTP uses a one-to-one mapping between Oblivious Relay and Gateway Resources; see Section 8.2 for more details.

This information allows the Client to send HTTP requests to the Oblivious Gateway Resource for forwarding to a Target Resource. The Oblivious Gateway Resource does not learn the Client's IP address or any other identifying information that might be revealed from the Client at the transport layer, nor does the Oblivious Gateway Resource learn which of the requests it receives are from the same Client.

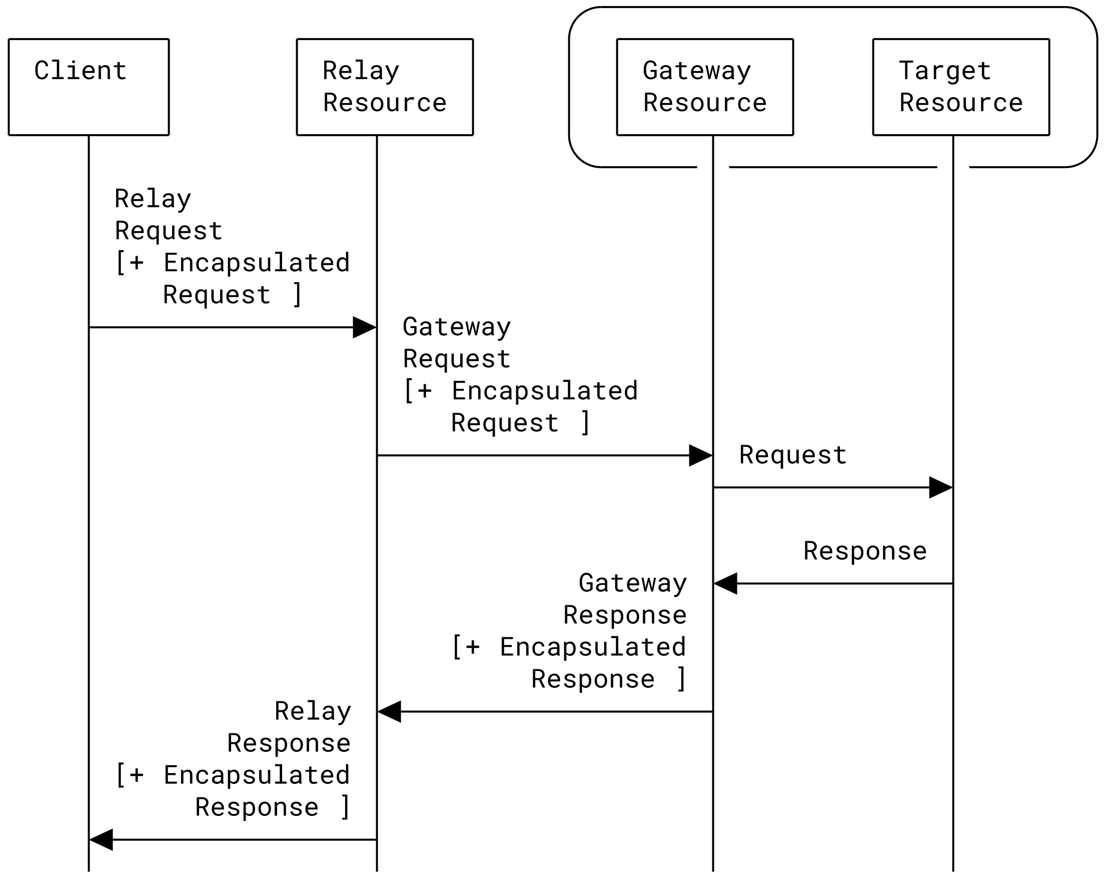
Client
Relay Resource
Gateway Resource
Target Resource
Relay Request [+ Encapsulated Request]
Gateway Request [+ Encapsulated Request]
Request
Response
Gateway Response [+ Encapsulated Response]
Relay Response [+ Encapsulated Response]

Figure 1: Overview of Oblivious HTTP

In order to forward a request for a Target Resource to the Oblivious Gateway Resource, the following steps occur, as shown in Figure 1:

1. The Client constructs an HTTP request for a Target Resource.
2. The Client encodes the HTTP request in a binary HTTP message and then encapsulates that message using HPKE and the process from Section 4.3.
3. The Client sends a POST request to the Oblivious Relay Resource with the Encapsulated Request as the content of that message.
4. The Oblivious Relay Resource forwards this request to the Oblivious Gateway Resource.
5. The Oblivious Gateway Resource receives this request and removes the HPKE protection to obtain an HTTP request.

The Oblivious Gateway Resource then handles the HTTP request. This typically involves making an HTTP request using the content of the Encapsulated Request. Once the Oblivious Gateway Resource has an HTTP response for this request, the following steps occur to return this response to the Client:

1. The Oblivious Gateway Resource encapsulates the HTTP response following the process in Section 4.4 and sends this in response to the request from the Oblivious Relay Resource.
2. The Oblivious Relay Resource forwards this response to the Client.
3. The Client removes the encapsulation to obtain the response to the original request.

This interaction provides authentication and confidentiality protection between the Client and the Oblivious Gateway, but importantly not between the Client and the Target Resource. While the Target Resource is a distinct HTTP resource from the Oblivious Gateway Resource, they are both logically under the control of the Oblivious Gateway, since the Oblivious Gateway Resource can unilaterally dictate the responses returned from the Target Resource to the Client. This arrangement is shown in Figure 1.

2.1. Applicability

Oblivious HTTP has limited applicability. Importantly, it requires explicit support from a willing Oblivious Relay Resource and Oblivious Gateway Resource, thereby limiting the use of Oblivious HTTP for generic applications; see Section 6.3 for more information.

Many uses of HTTP benefit from being able to carry state between requests, such as with cookies [COOKIES], authentication (Section 11 of [HTTP]), or even alternative services [RFC7838]. Oblivious HTTP removes linkage at the transport layer, which is only useful for an application that does not carry state between requests.

Oblivious HTTP is primarily useful where the privacy risks associated with possible stateful treatment of requests are sufficiently large that the cost of deploying this protocol can be justified. Oblivious HTTP is simpler and less costly than more robust systems, like Prio [PRIO] or Tor [DMS2004], which can provide stronger guarantees at higher operational costs.

Oblivious HTTP is more costly than a direct connection to a server. Some costs, like those involved with connection setup, can be amortized, but there are several ways in which Oblivious HTTP is more expensive than a direct request:

- Each request requires at least two regular HTTP requests, which could increase latency.
- Each request is expanded in size with additional HTTP fields, encryption-related metadata, and Authenticated Encryption with Associated Data (AEAD) expansion.
- Deriving cryptographic keys and applying them for request and response protection takes non-negligible computational resources.

Examples of where preventing the linking of requests might justify these costs include:

DNS queries:

DNS queries made to a recursive resolver reveal information about the requester, particularly if linked to other queries.

Telemetry submission: Applications that submit reports about their usage to their developers might use Oblivious HTTP for some types of moderately sensitive data.

These are examples of requests where there is information in a request that -- if it were connected to the identity of the user -- might allow a server to learn something about that user even if the identity of the user were pseudonymous. Other examples include submitting anonymous surveys, making search queries, or requesting location-specific content (such as retrieving tiles of a map display).

In addition to these limitations, Section 6 describes operational constraints that are necessary to realize the goals of the protocol.

2.2. 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.

This document uses terminology from [HTTP] and defines several terms as follows:

Client:
A Client originates Oblivious HTTP requests. A Client is also an HTTP client in two ways: for the Target Resource and for the Oblivious Relay Resource. However, when referring to the HTTP definition of client (Section 3.3 of [HTTP]), the term "HTTP client" is used; see Section 5.

Encapsulated Request:
An HTTP request that is encapsulated in an HPKE-encrypted message; see Section 4.3.

Encapsulated Response:
An HTTP response that is encapsulated in an HPKE-encrypted message; see Section 4.4.

Oblivious Relay Resource:
An intermediary that forwards Encapsulated Requests and Responses between Clients and a single Oblivious Gateway Resource. In context, this can be referred to simply as a "relay".

Oblivious Gateway Resource:
A resource that can receive an Encapsulated Request, extract the contents of that request, forward it to a Target Resource, receive a response, encapsulate that response, and then return the resulting Encapsulated Response. In context, this can be referred to simply as a "gateway".

Target Resource:
: The resource that is the target of an Encapsulated Request. This resource logically handles only regular HTTP requests and responses, so it might be ignorant of the use of Oblivious HTTP to reach it.

This document includes pseudocode that uses the functions and conventions defined in [HPKE].

Encoding an integer to a sequence of bytes in network byte order is described using the function `encode(n, v)`, where `n` is the number of bytes and `v` is the integer value. ASCII [ASCII] encoding of a string `s` is indicated using the function `encode_str(s)`.

Formats are described using notation from Section 1.3 of [QUIC]. An extension to that notation expresses the number of bits in a field using a simple mathematical function.

3. Key Configuration

A Client needs to acquire information about the key configuration of the Oblivious Gateway Resource in order to send Encapsulated Requests. In order to ensure that Clients do not encapsulate messages that other entities can intercept, the key configuration **MUST** be authenticated and have integrity protection.

This document does not define how that acquisition occurs. However, in order to help facilitate interoperability, it does specify a format for the keys. This ensures that different Client implementations can be configured in the same way and also enables advertising key configurations in a consistent format. This format might be used, for example, with HTTPS, as part of a system for configuring or discovering key configurations. However, note that such a system needs to consider the potential for key configuration to be used to compromise Client privacy; see Section 7.

A Client might have multiple key configurations to select from when encapsulating a request. Clients are responsible for selecting a preferred key configuration from those it supports. Clients need to consider both the Key Encapsulation Method (KEM) and the combinations of the Key Derivation Function (KDF) and AEAD in this decision.

3.1. Key Configuration Encoding

A single key configuration consists of a key identifier, a public key, an identifier for the KEM that the public key uses, and a set of HPKE symmetric algorithms. Each symmetric algorithm consists of an identifier for a KDF and an identifier for an AEAD.

Figure 2 shows a single key configuration.

```
HPKE Symmetric Algorithms {
  HPKE KDF ID (16),
  HPKE AEAD ID (16),
}

Key Config {
  Key Identifier (8),
  HPKE KEM ID (16),
  HPKE Public Key (Npk * 8),
  HPKE Symmetric Algorithms Length (16) = 4..65532,
  HPKE Symmetric Algorithms (32) ...,
}
```

Figure 2: A Single Key Configuration

That is, a key configuration consists of the following fields:

Key Identifier:
: An 8-bit value that identifies the key used by the Oblivious Gateway Resource.

HPKE KEM ID:
: A 16-bit value that identifies the KEM used for the identified key as defined in Section 7.1 of [HPKE] or the "HPKE KEM Identifiers" registry.

HPKE Public Key:
: The public key used by the gateway. The length of the public key is `Npk`, which is determined by the choice of HPKE KEM as defined in Section 4 of [HPKE].

HPKE Symmetric Algorithms Length:
: A 16-bit integer in network byte order that encodes the length, in bytes, of the HPKE Symmetric Algorithms field that follows.

HPKE Symmetric Algorithms:
: One or more pairs of identifiers for the different combinations of HPKE KDF and AEAD that the Oblivious Gateway Resource supports:

  HPKE KDF ID:
  : A 16-bit HPKE KDF identifier as defined in Section 7.2 of [HPKE] or the "HPKE KDF Identifiers" registry.

  HPKE AEAD ID:
  : A 16-bit HPKE AEAD identifier as defined in Section 7.3 of [HPKE] or the "HPKE AEAD Identifiers" registry.

3.2. Key Configuration Media Type

The "application/ohttp-keys" format is a media type that identifies a serialized collection of key configurations. The content of this media type comprises one or more key configuration encodings (see Section 3.1). Each encoded configuration is prefixed with a 2-byte integer in network byte order that indicates the length of the key configuration in bytes. The length-prefixed encodings are concatenated to form a list. See Section 9.1 for a definition of the media type.

Evolution of the key configuration format is supported through the definition of new formats that are identified by new media types.

A Client that receives an "application/ohttp-keys" object with encoding errors might be able to recover one or more key configurations. Differences in how key configurations are recovered might be exploited to segregate Clients, so Clients **MUST** discard incorrectly encoded key configuration collections.

4. HPKE Encapsulation

This document defines how a binary-encoded HTTP message [BINARY] is encapsulated using HPKE [HPKE]. Separate media types are defined to distinguish request and response messages:

- An Encapsulated Request format defined in Section 4.1 is identified by the "`message/ohttp-req`" media type (Section 9.2).
- An Encapsulated Response format defined in Section 4.2 is identified by the "`message/ohttp-res`" media type (Section 9.3).

Alternative encapsulations or message formats are indicated using the media type; see Sections 4.5 and 4.6.

4.1. Request Format

A message in "`message/ohttp-req`" format protects a binary HTTP request message; see Figure 3.

```
Request {
  Binary HTTP Message (..),
}
```

Figure 3: Plaintext Request Structure

This plaintext Request structure is encapsulated into a message in "`message/ohttp-req`" form by generating an Encapsulated Request. An Encapsulated Request comprises a key identifier; HPKE parameters for the chosen KEM, KDF, and AEAD; the encapsulated KEM shared secret (or `enc`); and an HPKE-protected binary HTTP request message.

An Encapsulated Request is shown in Figure 4. Section 4.3 describes the process for constructing and processing an Encapsulated Request.

```
Encapsulated Request {
  Key Identifier (8),
  HPKE KEM ID (16),
  HPKE KDF ID (16),
  HPKE AEAD ID (16),
  Encapsulated KEM Shared Secret (8 * Nenc),
  HPKE-Protected Request (..),
}
```

Figure 4: Encapsulated Request

That is, an Encapsulated Request comprises a Key Identifier, an HPKE KEM ID, an HPKE KDF ID, an HPKE AEAD ID, an Encapsulated KEM Shared Secret, and an HPKE-Protected Request. The Key Identifier, HPKE KEM ID, HPKE KDF ID, and HPKE AEAD ID fields are defined in Section 3.1. The Encapsulated KEM Shared Secret is the output of the `Encap()` function for the KEM, which is `Nenc` bytes in length, as defined in Section 4 of [HPKE].

4.2. Response Format

A message in "`message/ohttp-res`" format protects a binary HTTP response message; see Figure 5.

```
Response {
  Binary HTTP Message (..),
}
```

Figure 5: Plaintext Response Structure

This plaintext Response structure is encapsulated into a message in "`message/ohttp-res`" form by generating an Encapsulated Response. An Encapsulated Response comprises a nonce and the AEAD-protected binary HTTP response message.

An Encapsulated Response is shown in Figure 6. Section 4.4 describes the process for constructing and processing an Encapsulated Response.

```
Encapsulated Response {
  Nonce (8 * max(Nn, Nk)),
  AEAD-Protected Response (..),
}
```

Figure 6: Encapsulated Response

That is, an Encapsulated Response contains a Nonce and an AEAD-Protected Response. The Nonce field is either `Nn` or `Nk` bytes long, whichever is larger. The `Nn` and `Nk` values correspond to parameters of the AEAD used in HPKE, which is defined in Section 7.3 of [HPKE] or the "HPKE AEAD Identifiers" IANA registry. `Nn` and `Nk` refer to the size of the AEAD nonce and key, respectively, in bytes.

4.3. Encapsulation of Requests

Clients encapsulate a request, identified as `request`, using values from a key configuration:

- the key identifier from the configuration (`key_id`) with the corresponding KEM identified by `kem_id`,
- the public key from the configuration (`pkR`), and
- a combination of KDF (identified by `kdf_id`) and AEAD (identified by `aead_id`) that the Client selects from those in the key configuration.

The Client then constructs an Encapsulated Request, `enc_request`, from a binary-encoded HTTP request [BINARY] (`request`) as follows:

1. Construct a message header (`hdr`) by concatenating the values of `key_id`, `kem_id`, `kdf_id`, and `aead_id` as one 8-bit integer and three 16-bit integers, respectively, each in network byte order.
2. Build a sequence of bytes (`info`) by concatenating the ASCII-encoded string "message/bhttp request", a zero byte, and the header. Note: Section 4.6 discusses how alternative message formats might use a different `info` value.
3. Create a sending HPKE context by invoking `SetupBaseS()` (Section 5.1.1 of [HPKE]) with the public key of the receiver `pkR` and `info`. This yields the context `sctxt` and an encapsulation key `enc`.
4. Encrypt `request` by invoking the `Seal()` method on `sctxt` (Section 5.2 of [HPKE]) with empty associated data `aad`, yielding ciphertext `ct`.
5. Concatenate the values of `hdr`, `enc`, and `ct`. This yields an Encapsulated Request (`enc_request`).

Note that `enc` is of fixed length, so there is no ambiguity in parsing this structure.

In pseudocode, this procedure is as follows:

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

An Oblivious Gateway Resource decrypts an Encapsulated Request by reversing this process. To decapsulate an Encapsulated Request, `enc_request`:

1. Parse `enc_request` into `key_id`, `kem_id`, `kdf_id`, `aead_id`, `enc`, and `ct` (indicated using the function `parse()` in pseudocode). The Oblivious Gateway Resource is then able to find the HPKE private key, `skR`, corresponding to `key_id`.
   a. If `key_id` does not identify a key matching the type of `kem_id`, the Oblivious Gateway Resource returns an error.
   b. If `kdf_id` and `aead_id` identify a combination of KDF and AEAD that the Oblivious Gateway Resource is unwilling to use with `skR`, the Oblivious Gateway Resource returns an error.
2. Build a sequence of bytes (`info`) by concatenating the ASCII-encoded string "message/bhttp request"; a zero byte; `key_id` as an 8-bit integer; plus `kem_id`, `kdf_id`, and `aead_id` as three 16-bit integers.
3. Create a receiving HPKE context, `rctxt`, by invoking `SetupBaseR()` (Section 5.1.1 of [HPKE]) with `skR`, `enc`, and `info`.
4. Decrypt `ct` by invoking the `Open()` method on `rctxt` (Section 5.2 of [HPKE]), with an empty associated data `aad`, yielding `request` or an error on failure. If decryption fails, the Oblivious Gateway Resource returns an error.

In pseudocode, this procedure is as follows:

```
key_id, kem_id, kdf_id, aead_id, enc, ct = parse(enc_request)
info = concat(encode_str("message/bhttp request"),
              encode(1, 0),
              encode(1, key_id),
              encode(2, kem_id),
              encode(2, kdf_id),
              encode(2, aead_id))
rctxt = SetupBaseR(enc, skR, info)
request, error = rctxt.Open("", ct)
```

The Oblivious Gateway Resource retains the HPKE context, `rctxt`, so that it can encapsulate a response.

4.4. Encapsulation of Responses

Oblivious Gateway Resources generate an Encapsulated Response (`enc_response`) from a binary-encoded HTTP response [BINARY] (`response`). The Oblivious Gateway Resource uses the HPKE receiver context (`rctxt`) as the HPKE context (`context`) as follows:

1. Export a secret (`secret`) from `context`, using the string "message/bhttp response" as the `exporter_context` parameter to `context.Export`; see Section 5.3 of [HPKE]. The length of this secret is `max(Nn, Nk)`, where `Nn` and `Nk` are the length of the AEAD key and nonce that are associated with `context`. Note: Section 4.6 discusses how alternative message formats might use a different `context` value.

2. Generate a random value of length `max(Nn, Nk)` bytes, called `response_nonce`.
3. Extract a pseudorandom key (`prk`) using the `Extract` function provided by the KDF algorithm associated with `context`. The `ikm` input to this function is `secret`; the `salt` input is the concatenation of `enc` (from `enc_request`) and `response_nonce`.
4. Use the `Expand` function provided by the same KDF to create an AEAD key, `key`, of length `Nk` -- the length of the keys used by the AEAD associated with `context`. Generating `aead_key` uses a label of "key".
5. Use the same `Expand` function to create a nonce, `nonce`, of length `Nn` -- the length of the nonce used by the AEAD. Generating `aead_nonce` uses a label of "nonce".
6. Encrypt `response`, passing the AEAD function Seal the values of `aead_key`, `aead_nonce`, an empty `aad`, and a `pt` input of `response`. This yields `ct`.
7. Concatenate `response_nonce` and `ct`, yielding an Encapsulated Response, `enc_response`. Note that `response_nonce` is of fixed length, so there is no ambiguity in parsing either `response_nonce` or `ct`.

In pseudocode, this procedure is as follows:

```
secret = context.Export("message/bhttp response", max(Nn, Nk))
response_nonce = random(max(Nn, Nk))
salt = concat(enc, response_nonce)
prk = Extract(salt, secret)
aead_key = Expand(prk, "key", Nk)
aead_nonce = Expand(prk, "nonce", Nn)
ct = Seal(aead_key, aead_nonce, "", response)
enc_response = concat(response_nonce, ct)
```

Clients decrypt an Encapsulated Response by reversing this process. That is, Clients first parse `enc_response` into `response_nonce` and `ct`. Then, they follow the same process to derive values for `aead_key` and `aead_nonce`, using their sending HPKE context, `sctxt`, as the HPKE context, `context`.

The Client uses these values to decrypt `ct` using the AEAD function `Open`. Decrypting might produce an error, as follows:

```
response, error = Open(aead_key, aead_nonce, "", ct)
```

4.5. Request and Response Media Types

Media types are used to identify Encapsulated Requests and Responses; see Sections 9.2 and 9.3 for definitions of these media types.

Evolution of the format of Encapsulated Requests and Responses is supported through the definition of new formats that are identified by new media types. New media types might be defined to use a similar encapsulation with a different HTTP message format than in [BINARY]; see Section 4.6 for guidance on reusing this encapsulation method. Alternatively, a new encapsulation method might be defined.

4.6. Repurposing the Encapsulation Format

The encrypted payload of an Oblivious HTTP request and response is a binary HTTP message [BINARY]. The Client and Oblivious Gateway Resource agree on this encrypted payload type by specifying the media type "message/bhttp" in the HPKE info string and HPKE export context string for request and response encryption, respectively.

Future specifications may repurpose the encapsulation mechanism described in this document. This requires that the specification define a new media type. The encapsulation process for that content type can follow the same process, using new constant strings for the HPKE info and exporter context inputs.

For example, a future specification might encapsulate DNS messages, which use the "application/dns-message" media type [RFC8484]. In creating a new, encrypted media types, specifications might define the use of string "application/dns-message request" (plus a zero byte and the header for the full value) for request encryption and the string "application/dns-message response" for response encryption.

5. HTTP Usage

A Client interacts with the Oblivious Relay Resource by constructing an Encapsulated Request. This Encapsulated Request is included as the content of a POST request to the Oblivious Relay Resource. This request only needs those fields necessary to carry the Encapsulated Request: a method of POST, a target URI of the Oblivious Relay Resource, a header field containing the content type (see Section 9.2), and the Encapsulated Request as the request content. In the request to the Oblivious Relay Resource, Clients **MAY** include additional fields. However, additional fields **MUST** be independent of the Encapsulated Request and **MUST** be fields that the Oblivious Relay Resource will remove before forwarding the Encapsulated Request towards the target, such as the `Connection` or `Proxy-Authorization` header fields [HTTP].

The Client role in this protocol acts as an HTTP client both with respect to the Oblivious Relay Resource and the Target Resource. The request, which the Client makes to the Target Resource, diverges from typical HTTP assumptions about the use of a connection (see Section 3.3 of [HTTP]) in that the request and response are encrypted rather than sent over a connection. The Oblivious Relay Resource and the Oblivious Gateway Resource also act as HTTP clients toward the Oblivious Gateway Resource and Target Resource, respectively.

In order to achieve the privacy and security goals of the protocol, a Client also needs to observe the guidance in Section 6.1.

The Oblivious Relay Resource interacts with the Oblivious Gateway Resource as an HTTP client by constructing a request using the same restrictions as the Client request, except that the target URI is the Oblivious Gateway Resource. The content of this request is copied from the Client. An Oblivious Relay Resource **MAY** reject requests that are obviously invalid, such as a request with no content. The Oblivious Relay Resource **MUST NOT** add information to the request without the Client being aware of the type of information that might be added; see Section 6.2 for more information on relay responsibilities.

When a response is received from the Oblivious Gateway Resource, the Oblivious Relay Resource forwards the response according to the rules of an HTTP proxy; see Section 7.6 of [HTTP]. In case of timeout or error, the Oblivious Relay Resource can generate a response with an appropriate status code.

In order to achieve the privacy and security goals of the protocol, an Oblivious Relay Resource also needs to observe the guidance in Section 6.2.

An Oblivious Gateway Resource acts as a gateway for requests to the Target Resource (see Section 7.6 of [HTTP]). The one exception is that any information it might forward in a response **MUST** be encapsulated, unless it is responding to errors that do not relate to processing the contents of the Encapsulated Request; see Section 5.2.

An Oblivious Gateway Resource, if it receives any response from the Target Resource, sends a single 200 response containing the Encapsulated Response. Like the request from the Client, this response **MUST** only contain those fields necessary to carry the Encapsulated Response: a 200 status code, a header field indicating the content type, and the Encapsulated Response as the response content. As with requests, additional fields **MAY** be used to convey information that does not reveal information about the Encapsulated Response.

An Oblivious Gateway Resource that does not receive a response can itself generate a response with an appropriate error status code (such as 504 (Gateway Timeout); see Section 15.6.5 of [HTTP]), which is then encapsulated in the same way as a successful response.

In order to achieve the privacy and security goals of the protocol, an Oblivious Gateway Resource also needs to observe the guidance in Section 6.3.

5.1. Informational Responses

This encapsulation does not permit progressive processing of responses. Though the binary HTTP response format does support the inclusion of informational (1xx) status codes, the AEAD encapsulation cannot be removed until the entire message is received.

In particular, the `Expect` header field with 100-continue (see Section 10.1.1 of [HTTP]) cannot be used. Clients **MUST NOT** construct a request that includes a 100-continue expectation; the Oblivious Gateway Resource **MUST** generate an error if a 100-continue expectation is received.

5.2. Errors

A server that receives an invalid message for any reason **MUST** generate an HTTP response with a 4xx status code.

Errors detected by the Oblivious Relay Resource and errors detected by the Oblivious Gateway Resource before removing protection (including being unable to remove encapsulation for any reason) result in the status code being sent without protection in response to the POST request made to that resource.

Errors detected by the Oblivious Gateway Resource after successfully removing encapsulation and errors detected by the Target Resource **MUST** be sent in an Encapsulated Response. This might be because the Encapsulated Request is malformed or the Target Resource does not produce a response. In either case, the Oblivious Gateway Resource can generate a response with an appropriate error status code (such as 400 (Bad Request) or 504 (Gateway Timeout); see Sections 15.5.1 and 15.6.5 of [HTTP], respectively). This response is encapsulated in the same way as a successful response.

Errors in the encapsulation of requests mean that responses cannot be encapsulated. This includes cases where the key configuration is incorrect or outdated. The Oblivious Gateway Resource can generate and send a response with a 4xx status code to the Oblivious Relay Resource. This response **MAY** be forwarded to the Client or treated by the Oblivious Relay Resource as a failure. If a Client receives a response that is not an Encapsulated Response, this could indicate that the Client configuration used to construct the request is incorrect or out of date.

5.3. Signaling Key Configuration Problems

The problem type [PROBLEM] of "https://iana.org/assignments/http-problem-types#ohttp-key" is defined in this section. An Oblivious Gateway Resource **MAY** use this problem type in a response to indicate that an Encapsulated Request used an outdated or incorrect key configuration.

Figure 7 shows an example response in HTTP/1.1 format.

```
HTTP/1.1 400 Bad Request
Date: Mon, 07 Feb 2022 00:28:05 GMT
Content-Type: application/problem+json
Content-Length: 106

{"type":"https://iana.org/assignments/http-problem-types#ohttp-key",
"title": "key identifier unknown"}
```

Figure 7: Example Rejection of Key Configuration

As this response cannot be encrypted, it might not reach the Client. A Client cannot rely on the Oblivious Gateway Resource using this problem type. A Client might also be configured to disregard responses that are not encapsulated on the basis that they might be subject to observation or modification by an Oblivious Relay Resource. A Client might manage the risk of an outdated key configuration using a heuristic approach whereby it periodically refreshes its key configuration if it receives a response with an error status code that has not been encapsulated.

6. Security Considerations

In this design, a Client wishes to make a request to an Oblivious Gateway Resource that is forwarded to a Target Resource. The Client wishes to make this request without linking that request with either of the following:

- The identity at the network and transport layer of the Client (that is, the Client IP address and TCP or UDP port number the Client uses to create a connection).
- Any other request the Client might have made in the past or might make in the future.

In order to ensure this, the Client selects a relay (that serves the Oblivious Relay Resource) that it trusts will protect this information by forwarding the Encapsulated Request and Response without passing it to the server (that serves the Oblivious Gateway Resource).

In this section, a deployment where there are three entities is considered:

- A Client makes requests and receives responses.
- A relay operates the Oblivious Relay Resource.
- A server operates both the Oblivious Gateway Resource and the Target Resource.

Section 6.10 discusses the security implications for a case where different servers operate the Oblivious Gateway Resource and Target Resource.

Requests from the Client to Oblivious Relay Resource and from Oblivious Relay Resource to Oblivious Gateway Resource **MUST** use HTTPS in order to provide unlinkability in the presence of a network observer.

To achieve the stated privacy goals, the Oblivious Relay Resource cannot be operated by the same entity as the Oblivious Gateway Resource. However, colocation of the Oblivious Gateway Resource and Target Resource simplifies the interactions between those resources without affecting Client privacy.

As a consequence of this configuration, Oblivious HTTP prevents linkability described above. Informally, this means:

1. Requests and responses are known only to Clients and Oblivious Gateway Resources. In particular, the Oblivious Relay Resource knows the origin and destination of an Encapsulated Request and Response, yet it does not know the decrypted contents. Likewise, Oblivious Gateway Resources learn only the Oblivious Relay Resource and the decrypted request. No entity other than the Client can see the plaintext request and response and can attribute them to the Client.

2. Oblivious Gateway Resources, and therefore Target Resources, cannot link requests from the same Client in the absence of unique per-Client keys.

Traffic analysis that might affect these properties is outside the scope of this document; see Section 6.2.3.

A formal analysis of Oblivious HTTP is in [OHTTP-ANALYSIS].

6.1. Client Responsibilities

Because Clients do not authenticate the Target Resource when using Oblivious HTTP, Clients **MUST** have some mechanism to authorize an Oblivious Gateway Resource for use with a Target Resource. One possible means of authorization is an allowlist. This ensures that Oblivious Gateway Resources are not misused to forward traffic to arbitrary Target Resources. Section 6.3 describes similar responsibilities that apply to Oblivious Gateway Resources.

Clients **MUST** ensure that the key configuration they select for generating Encapsulated Requests is integrity protected and authenticated so that it can be attributed to the Oblivious Gateway Resource; see Section 3.

Since Clients connect directly to the Oblivious Relay Resource instead of the Target Resource, application configurations wherein Clients make policy decisions about target connections, e.g., to apply certificate pinning, are incompatible with Oblivious HTTP. In such cases, alternative technologies such as HTTP CONNECT (Section 9.3.6 of [HTTP]) can be used. Applications could implement related policies on key configurations and relay connections, though these might not provide the same properties as policies enforced directly on target connections. Instead, when this difference is relevant, applications can connect directly to the target at the cost of either privacy or performance.

Clients cannot carry connection-level state between requests as they only establish direct connections to the relay responsible for the Oblivious Relay Resource. However, the content of requests might be used by a server to correlate requests. Cookies [COOKIES] are the most obvious feature that might be used to correlate requests, but any identity information and authentication credentials might have the same effect. Clients also need to treat information learned from responses with similar care when constructing subsequent requests, which includes the identity of resources.

Clients **MUST** generate a new HPKE context for every request, using a good source of entropy [RANDOM] for generating keys. Key reuse not only risks requests being linked but also could expose request and response contents to the relay.

The request the Client sends to the Oblivious Relay Resource only requires minimal information; see Section 5. The request that carries the Encapsulated Request and that is sent to the Oblivious Relay Resource **MUST NOT** include identifying information unless the Client can trust that this information is removed by the relay. A Client **MAY** include information only for the Oblivious Relay Resource in header fields identified by the `Connection` header field if it trusts the relay to remove these, as required by Section 7.6.1 of [HTTP]. The Client needs to trust that the relay does not replicate the source addressing information in the request it forwards.

Clients rely on the Oblivious Relay Resource to forward Encapsulated Requests and Responses. However, the relay can only refuse to forward messages; it cannot inspect or modify the contents of Encapsulated Requests or Responses.

6.2. Relay Responsibilities

The relay that serves the Oblivious Relay Resource has a very simple function to perform. For each request it receives, it makes a request of the Oblivious Gateway Resource that includes the same content. When it receives a response, it sends a response to the Client that includes the content of the response from the Oblivious Gateway Resource.

When forwarding a request, the relay **MUST** follow the forwarding rules in Section 7.6 of [HTTP]. A generic HTTP intermediary implementation is suitable for the purposes of serving an Oblivious Relay Resource, but additional care is needed to ensure that Client privacy is maintained.

Firstly, a generic implementation will forward unknown fields. For Oblivious HTTP, an Oblivious Relay Resource **SHOULD NOT** forward unknown fields. Though Clients are not expected to include fields that might contain identifying information, removing unknown fields removes this privacy risk.

Secondly, generic implementations are often configured to augment requests with information about the Client, such as the Via field or the Forwarded field [FORWARDED]. A relay **MUST NOT** add information when forwarding requests that might be used to identify Clients, except for information that a Client is aware of; see Section 6.2.1.

Finally, a relay can also generate responses, though it is assumed to not be able to examine the content of a request (other than to observe the choice of key identifier, KDF, and AEAD); therefore, it is also assumed that it cannot generate an Encapsulated Response.

6.2.1. Differential Treatment

A relay **MAY** add information to requests if the Client is aware of the nature of the information that could be added. Any addition **MUST NOT** include information that uniquely and permanently identifies the Client, including any pseudonymous identifier. Information added by the relay -- beyond what is already revealed through Encapsulated Requests from Clients -- can reduce the size of the anonymity set of Clients at a gateway.

A Client does not need to be aware of the exact value added for each request but needs to know the range of possible values the relay might use. How a Client might learn about added information is not defined in this document.

Moreover, relays **MAY** apply differential treatment to Clients that engage in abusive behavior, e.g., by sending too many requests in comparison to other Clients, or as a response to rate limits signaled from the gateway. Any such differential treatment can reveal information to the gateway that would not be revealed otherwise and therefore reduce the size of the anonymity set of Clients using a gateway. For example, if a relay chooses to rate limit or block an abusive Client, this means that any Client requests that are not treated this way are known to be non-abusive by the gateway. Clients need to consider the likelihood of such differential treatment and the privacy risks when using a relay.

Some patterns of abuse cannot be detected without access to the request that is made to the target. This means that only the gateway or the target is in a position to identify abuse. A gateway **MAY** send signals toward the relay to provide feedback about specific requests. For example, a gateway could respond differently to requests it cannot decapsulate, as mentioned in Section 5.2. A relay that acts on this feedback could -- either inadvertently or by design -- lead to Client deanonymization.

6.2.2. Denial of Service

As there are privacy benefits from having a large rate of requests forwarded by the same relay (see Section 6.2.3), servers that operate the Oblivious Gateway Resource might need an arrangement with Oblivious Relay Resources. This arrangement might be necessary to prevent having the large volume of requests being classified as an attack by the server.

If a server accepts a larger volume of requests from a relay, it needs to trust that the relay does not allow abusive levels of request volumes from Clients. That is, if a server allows requests from the relay to be exempt from rate limits, the server might want to ensure that the relay applies a rate-limiting policy that is acceptable to the server.

Servers that enter into an agreement with a relay that enables a higher request rate might choose to authenticate the relay to enable the higher rate.

6.2.3. Traffic Analysis

Using HTTPS protects information about which resources are the subject of request and prevents a network observer from being able to trivially correlate messages on either side of a relay. However, using HTTPS does not prevent traffic analysis by such network observers.

The time at which Encapsulated Request or Response messages are sent can reveal information to a network observer. Though messages exchanged between the Oblivious Relay Resource and the Oblivious Gateway Resource might be sent in a single connection, traffic analysis could be used to match messages that are forwarded by the relay.

A relay could, as part of its function, delay requests before forwarding them. Delays might increase the anonymity set into which each request is attributed. Any delay also increases the time that a Client waits for a response, so delays **SHOULD** only be added with the consent -- or at least awareness -- of Clients.

A relay that forwards large volumes of exchanges can provide better privacy by providing larger sets of messages that need to be matched.

Traffic analysis is not restricted to network observers. A malicious Oblivious Relay Resource could use traffic analysis to learn information about otherwise encrypted requests and responses relayed between Clients and gateways. An Oblivious Relay Resource terminates TLS connections from Clients, so they see message boundaries. This privileged position allows for richer feature extraction from encrypted data, which might improve traffic analysis.

Clients and Oblivious Gateway Resources can use padding to reduce the effectiveness of traffic analysis. Padding is a capability provided by binary HTTP messages; see Section 3.8 of [BINARY]. If the encapsulation method described in this document is used to protect a different message type (see Section 4.6), that message format might need to include padding support. Oblivious Relay Resources can also use padding for the same reason but need to operate at the HTTP layer since they cannot manipulate binary HTTP messages; for example, see Section 10.7 of [HTTP/2] or Section 10.7 of [HTTP/3]).

6.3. Server Responsibilities

The Oblivious Gateway Resource can be operated by a different entity than the Target Resource. However, this means that the Client needs to trust the Oblivious Gateway Resource not to modify requests or responses. This analysis concerns itself with a deployment scenario where a single server provides both the Oblivious Gateway Resource and Target Resource.

A server that operates both Oblivious Gateway and Target Resources is responsible for removing request encryption, generating a response to the Encapsulated Request, and encrypting the response.

Servers should account for traffic analysis based on response size or generation time. Techniques such as padding or timing delays can help protect against such attacks; see Section 6.2.3.

If separate entities provide the Oblivious Gateway Resource and Target Resource, these entities might need an arrangement similar to that between server and relay for managing denial of service; see Section 6.2.2. Moreover, the Oblivious Gateway Resource **SHOULD** have some mechanism to ensure that the Oblivious Gateway Resource is not misused as a relay for HTTP messages to an arbitrary Target Resource, such as an allowlist.

Non-secure requests -- such as those with the "http" scheme as opposed to the "https" scheme -- **SHOULD NOT** be used if the Oblivious Gateway and Target Resources are not on the same origin. If messages are forwarded between these resources without the protections afforded by HTTPS, they could be inspected or modified by a network attacker. Note that a request could be forwarded without protection if the two resources share an origin.

6.4. Key Management

An Oblivious Gateway Resource needs to have a plan for replacing keys. This might include regular replacement of keys, which can be assigned new key identifiers. If an Oblivious Gateway Resource receives a request that contains a key identifier that it does not understand or that corresponds to a key that has been replaced, the server can respond with an HTTP 422 (Unprocessable Content) status code.

A server can also use a 422 status code if the server has a key that corresponds to the key identifier, but the Encapsulated Request cannot be successfully decrypted using the key.

A server **MUST** ensure that the HPKE keys it uses are not valid for any other protocol that uses HPKE with the "message/bhttp request" label. Designers of protocols that reuse this encryption format, especially new versions of this protocol, can ensure key diversity by choosing a different label in their use of HPKE. The "message/bhttp response" label was chosen for symmetry only as it provides key diversity only within the HPKE context created using the "message/bhttp request" label; see Section 4.6.

6.5. Replay Attacks

A server is responsible for either rejecting replayed requests or ensuring that the effect of replays does not adversely affect Clients or resources.

Encapsulated Requests can be copied and replayed by the Oblivious Relay Resource. The threat model for Oblivious HTTP allows the possibility that an Oblivious Relay Resource might replay requests. Furthermore, if a Client sends an Encapsulated Request in TLS early data (see Section 8 of [TLS] and [RFC8470]), a network-based adversary might be able to cause the request to be replayed. In both cases, the effect of a replay attack and the mitigations that might be employed are similar to TLS early data.

It is the responsibility of the application that uses Oblivious HTTP to either reject replayed requests or ensure that replayed requests have no adverse effect on their operation. This section describes some approaches that are universally applicable and suggestions for more targeted techniques.

A Client or Oblivious Relay Resource **MUST NOT** automatically attempt to retry a failed request unless it receives a positive signal indicating that the request was not processed or forwarded. The HTTP/2 REFUSED_STREAM error code (Section 8.1.4 of [HTTP/2]), the HTTP/3 H3_REQUEST_REJECTED error code (Section 8.1 of [HTTP/3]), or a GOAWAY frame with a low enough identifier (in either protocol version) are all sufficient signals that no processing occurred. HTTP/1.1 [HTTP/1.1] provides no equivalent signal. Connection failures or interruptions are not sufficient signals that no processing occurred.

The anti-replay mechanisms described in Section 8 of [TLS] are generally applicable to Oblivious HTTP requests. The encapsulated keying material (or `enc`) can be used in place of a nonce to uniquely identify a request. This value is a high-entropy value that is freshly generated for every request, so two valid requests will have different values with overwhelming probability.

The mechanism used in TLS for managing differences in Client and server clocks cannot be used as it depends on being able to observe previous interactions. Oblivious HTTP explicitly prevents such linkability.

The considerations in [RFC8470] as they relate to managing the risk of replay also apply, though there is no option to delay the processing of a request.

Limiting requests to those with safe methods might not be satisfactory for some applications, particularly those that involve the submission of data to a server. The use of idempotent methods might be of some use in managing replay risk, though it is important to recognize that different idempotent requests can be combined to be not idempotent.

Even without replay prevention, the server-chosen `response_nonce` field ensures that responses have unique AEAD keys and nonces even when requests are replayed.

6.5.1. Use of Date for Anti-replay

Clients **SHOULD** include a `Date` header field in Encapsulated Requests, unless the Client has prior knowledge that indicates that the Oblivious Gateway Resource does not use `Date` for anti-replay purposes.

Though HTTP requests often do not include a `Date` header field, the value of this field might be used by a server to limit the amount of requests it needs to track if it needs to prevent replay attacks.

An Oblivious Gateway Resource can maintain state for requests for a small window of time over which it wishes to accept requests. The Oblivious Gateway Resource can store all requests it processes within this window. Storing just the `enc` field of a request, which should be unique to each request, is sufficient. The Oblivious Gateway Resource can reject any request that is the same as one that was previously answered within that time window or if the `Date` header field from the decrypted request is outside of the current time window.

Oblivious Gateway Resources might need to allow for the time it takes requests to arrive from the Client, with a time window that is large enough to allow for differences in clocks. Insufficient tolerance of time differences could result in valid requests being unnecessarily rejected. Beyond allowing for multiple round-trip times -- to account for retransmission -- network delays are unlikely to be significant in determining the size of the window, unless all potential Clients are known to have excellent timekeeping. A specific window size might need to be determined experimentally.

Oblivious Gateway Resources **MUST NOT** treat the time window as secret information. An attacker can actively probe with different values for the `Date` field to determine the time window over which the server will accept responses.

6.5.2. Correcting Clock Differences

An Oblivious Gateway Resource can reject requests that contain a `Date` value that is outside of its active window with a 400 series status code. The problem type [PROBLEM] of "https://iana.org/assignments/http-problem-types#date" is defined to allow the server to signal that the `Date` value in the request was unacceptable.

Figure 8 shows an example response in HTTP/1.1 format.

```
HTTP/1.1 400 Bad Request
Date: Mon, 07 Feb 2022 00:28:05 GMT
Content-Type: application/problem+json
Content-Length: 128

{"type":"https://iana.org/assignments/http-problem-types#date",
"title": "date field in request outside of acceptable range"}
```

Figure 8: Example Rejection of Request Date Field

Disagreements about time are unlikely if both Client and Oblivious Gateway Resource have a good source of time; see [NTP]. However, clock differences are known to be commonplace; see Section 7.1 of [CLOCKSKEW].

Including a `Date` header field in the response allows the Client to correct clock errors by retrying the same request using the value of the `Date` field provided by the Oblivious Gateway Resource. The value of the `Date` field can be copied if the response is fresh, with an adjustment based on the `Age` field otherwise; see Section 4.2 of [HTTP-CACHING]. When retrying a request, the Client **MUST** create a fresh encryption of the modified request, using a new HPKE context.

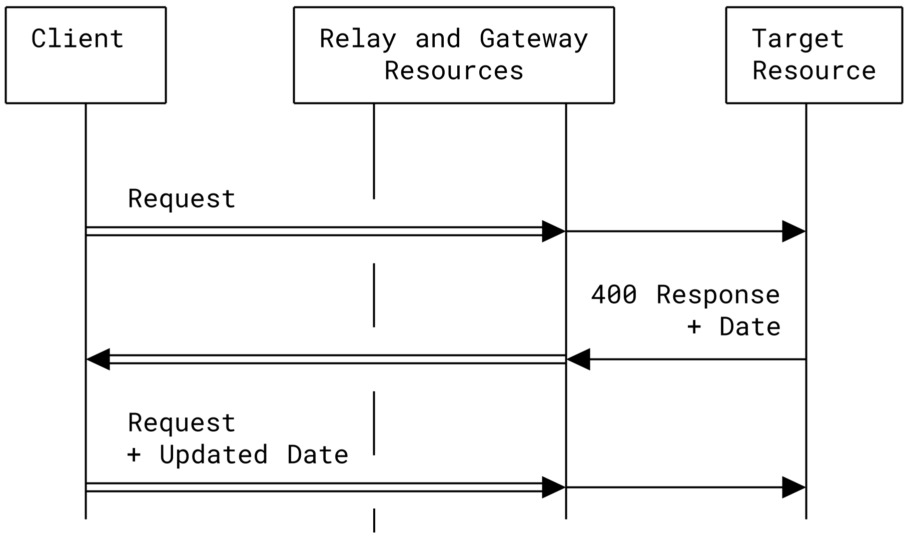

Figure 9: Retrying with an Updated Date Field

Retrying immediately allows the Oblivious Gateway Resource to measure the round-trip time to the Client. The observed delay might reveal something about the location of the Client. Clients could delay retries to add some uncertainty to any observed delay.

Intermediaries can sometimes rewrite the `Date` field when forwarding responses. This might cause problems if the Oblivious Gateway Resource and intermediary clocks differ by enough to cause the retry to be rejected. Therefore, Clients **MUST NOT** retry a request with an adjusted date more than once.

Oblivious Gateway Resources that condition their responses on the `Date` header field **SHOULD** either ensure that intermediaries do not cache responses (by including a `Cache-Control` directive of `no-store`) or designate the response as conditional on the value of the `Date` request header field (by including the token "date" in a `Vary` header field).

Clients **MUST NOT** use the date provided by the Oblivious Gateway Resource for any other purpose, including future requests to any resource. Any request that uses information provided by the Oblivious Gateway Resource might be correlated using that information.

6.6. Forward Secrecy

This document does not provide forward secrecy for either requests or responses during the lifetime of the key configuration. A measure of forward secrecy can be provided by generating a new key configuration then deleting the old keys after a suitable period.

6.7. Post-Compromise Security

This design does not provide post-compromise security for responses.

A Client only needs to retain keying material that might be used to compromise the confidentiality and integrity of a response until that response is consumed, so there is negligible risk associated with a Client compromise.

A server retains a secret key that might be used to remove protection from messages over much longer periods. A server compromise that provided access to the Oblivious Gateway Resource secret key could allow an attacker to recover the plaintext of all requests sent toward affected keys and all of the responses that were generated.

Even if server keys are compromised, an adversary cannot access messages exchanged by the Client with the Oblivious Relay Resource as messages are protected by TLS. Use of a compromised key also requires that the Oblivious Relay Resource cooperate with the attacker or that the attacker is able to compromise these TLS connections.

The total number of messages affected by server key compromise can be limited by regular rotation of server keys.

6.8. Client Clock Exposure

Including a `Date` field in requests reveals some information about the Client clock. This might be used to fingerprint Clients [UWT] or to identify Clients that are vulnerable to attacks that depend on incorrect clocks.

Clients can randomize the value that they provide for `Date` to obscure the true value of their clock and reduce the chance of linking requests over time. However, this increases the risk that their request is rejected as outside the acceptable window.

6.9. Media Type Security

The key configuration media type defined in Section 3.2 represents keying material. The content of this media type is not active (see Section 4.6 of [RFC6838]), but it governs how a Client might interact with an Oblivious Gateway Resource. The security implications of processing it are described in Section 6.1; privacy implications are described in Section 7.

The security implications of handling the message media types defined in Section 4.5 is covered in other parts of this section in more detail. However, these message media types are also encrypted encapsulations of HTTP requests and responses.

HTTP messages contain content, which can use any media type. In particular, requests are processed by an Oblivious Target Resource, which -- as an HTTP resource -- defines how content is processed; see Section 3.1 of [HTTP]. HTTP clients can also use resource identity and response content to determine how content is processed. Consequently, the security considerations of Section 17 of [HTTP] also apply to the handling of the content of these media types.

6.10. Separate Gateway and Target

This document generally assumes that the same entity operates the Oblivious Gateway Resource and the Target Resource. However, as the Oblivious Gateway Resource performs generic HTTP processing, the use of forwarding cannot be completely precluded.

The scheme specified in the Encapsulated Request determines the security requirements for any protocol that is used between the Oblivious Gateway and Target Resources. Using HTTPS is **RECOMMENDED**; see Section 6.3.

A Target Resource that is operated on a different server from the Oblivious Gateway Resource is an ordinary HTTP resource. A Target Resource can privilege requests that are forwarded by a given Oblivious Gateway Resource if it trusts the operator of the Oblivious Gateway Resource to only forward requests that meet the expectations of the Target Resource. Otherwise, the Target Resource treats requests from an Oblivious Gateway Resource no differently than any other HTTP client.

For instance, an Oblivious Gateway Resource might -- possibly with the help of Oblivious Relay Resources -- be trusted not to forward an excessive volume of requests. This might allow the Target Resource to accept a greater volume of requests from that Oblivious Gateway Resource relative to other HTTP clients.

An Oblivious Gateway Resource could implement policies that improve the ability of the Target Resource to implement policy exemptions, such as only forwarding requests toward specific Target Resources according to an allowlist; see Section 6.3.

7. Privacy Considerations

One goal of this design is that independent Client requests are only linkable by their content. However, the choice of Client configuration might be used to correlate requests. A Client configuration includes the Oblivious Relay Resource URI, the Oblivious Gateway key configuration, and the Oblivious Gateway Resource URI. A configuration is active if Clients can successfully use it for interacting with a target.

Oblivious Relay and Gateway Resources can identify when requests use the same configuration by matching the key identifier from the key configuration or the Oblivious Gateway Resource URI. The Oblivious Gateway Resource might use the source address of requests to correlate requests that use an Oblivious Relay Resource run by the same operator. If the Oblivious Gateway Resource is willing to use trial decryption, requests can be further separated into smaller groupings based on active configurations that clients use.

Each active Client configuration partitions the Client anonymity set. In practice, it is infeasible to reduce the number of active configurations to one. Enabling diversity in choice of Oblivious Relay Resource naturally increases the number of active configurations. More than one configuration might need to be active to allow for key rotation and server maintenance.

Client privacy depends on having each configuration used by many other Clients. It is critical to prevent the use of unique Client configurations, which might be used to track individual Clients, but it is also important to avoid creating small groupings of Clients that might weaken privacy protections.

A specific method for a Client to acquire configurations is not included in this specification. Applications using this design **MUST** provide accommodations to mitigate tracking using Client configurations. [CONSISTENCY] provides options for ensuring that Client configurations are consistent between Clients.

The content of requests or responses, if used in forming new requests, can be used to correlate requests. This includes obvious methods of linking requests, like cookies [COOKIES], but it also includes any information in either message that might affect how subsequent requests are formulated. For example, [FIELDING] describes how interactions that are individually stateless can be used to build a stateful system when a Client acts on the content of a response.

8. Operational and Deployment Considerations

This section discusses various operational and deployment considerations.

8.1. Performance Overhead

Using Oblivious HTTP adds both cryptographic overhead and latency to requests relative to a simple HTTP request-response exchange. Deploying relay services that are on path between Clients and servers avoids adding significant additional delay due to network topology. A study of a similar system [ODOH-PETS] found that deploying proxies close to servers was most effective in minimizing additional latency.

8.2. Resource Mappings

This protocol assumes a fixed, one-to-one mapping between the Oblivious Relay Resource and the Oblivious Gateway Resource. This means that any Encapsulated Request sent to the Oblivious Relay Resource will always be forwarded to the Oblivious Gateway Resource. This constraint was imposed to simplify relay configuration and mitigate against the Oblivious Relay Resource being used as a generic relay for unknown Oblivious Gateway Resources. The relay will only forward for Oblivious Gateway Resources that it has explicitly configured and allowed.

It is possible for a server to be configured with multiple Oblivious Relay Resources, each for a different Oblivious Gateway Resource as needed. If the goal is to support a large number of Oblivious Gateway Resources, Clients might be provided with a URI template [TEMPLATE], from which multiple Oblivious Relay Resources could be constructed.

8.3. Network Management

Oblivious HTTP might be incompatible with network interception regimes, such as those that rely on configuring Clients with trust anchors and intercepting TLS connections. While TLS might be intercepted successfully, interception middlebox devices might not receive updates that would allow Oblivious HTTP to be correctly identified using the media types defined in Sections 9.2 and 9.3.

Oblivious HTTP has a simple key management design that is not trivially altered to enable interception by intermediaries. Clients that are configured to enable interception might choose to disable Oblivious HTTP in order to ensure that content is accessible to middleboxes.

9. IANA Considerations

IANA has registered the following media types in the "Media Types" registry at <https://iana.org/assignments/media-types>, following the procedures of [RFC6838]: "`application/ohttp-keys`" (Section 9.1), "`message/ohttp-req`" (Section 9.2), and "`message/ohttp-res`" (Section 9.3).

IANA has added the following types to the "HTTP Problem Types" registry at <https://iana.org/assignments/http-problem-types>: "date" (Section 9.4) and "ohttp-key" (Section 9.5).

9.1. application/ohttp-keys Media Type

The "`application/ohttp-keys`" media type identifies a key configuration used by Oblivious HTTP.

Type name: application

Subtype name: ohttp-keys

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: "binary"

Security considerations: See Section 6.9

Interoperability considerations: N/A

Published specification: RFC 9458

Applications that use this media type: This type identifies a key configuration as used by Oblivious HTTP and applications that use Oblivious HTTP.

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

9.2. message/ohttp-req Media Type

The "`message/ohttp-req`" identifies an encrypted binary HTTP request. This is a binary format that is defined in Section 4.3.

Type name: message

Subtype name: ohttp-req

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: "binary"

Security considerations: See Section 6.9

Interoperability considerations: N/A

Published specification: RFC 9458

Applications that use this media type: Oblivious HTTP and applications that use Oblivious HTTP use this media type to identify encapsulated binary HTTP requests.

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

9.3. message/ohttp-res Media Type

The "`message/ohttp-res`" identifies an encrypted binary HTTP response. This is a binary format that is defined in Section 4.4.

Type name: message

Subtype name: ohttp-res

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: "binary"

Security considerations: See Section 6.9

Interoperability considerations: N/A

Published specification: RFC 9458

Applications that use this media type: Oblivious HTTP and applications that use Oblivious HTTP use this media type to identify encapsulated binary HTTP responses.

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

9.4. Registration of "date" Problem Type

IANA has added a new entry in the "HTTP Problem Types" registry established by [PROBLEM].

Type URI: https://iana.org/assignments/http-problem-types#date

Title: Date Not Acceptable

Recommended HTTP Status Code: 400

Reference: Section 6.5.2 of RFC 9458

9.5. Registration of "ohttp-key" Problem Type

IANA has added a new entry in the "HTTP Problem Types" registry established by [PROBLEM].

Type URI: https://iana.org/assignments/http-problem-types#ohttp-key

Title: Oblivious HTTP key configuration not acceptable

Recommended HTTP Status Code: 400

Reference: Section 5.3 of RFC 9458

10. References

10.1. Normative References

[ASCII] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[BINARY] Thomson, M. and C. A. Wood, "Binary Representation of HTTP Messages", RFC 9292, DOI 10.17487/RFC9292, August 2022, <https://www.rfc-editor.org/info/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/info/rfc9180>.

[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/info/rfc9110>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[PROBLEM] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

[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/info/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/info/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/info/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/info/rfc8174>.

[RFC8470] Thomson, M., Nottingham, M., and W. Tarreau, "Using Early Data in HTTP", RFC 8470, DOI 10.17487/RFC8470, September 2018, <https://www.rfc-editor.org/info/rfc8470>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

10.2. Informative References

[CLOCKSKEW] Acer, M., Stark, E., Felt, A., Fahl, S., Bhargava, R., Dev, B., Braithwaite, M., Sleevi, R., and P. Tabriz, "Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, DOI 10.1145/3133956.3134007, October 2017, <https://doi.org/10.1145/3133956.3134007>.

[CONSISTENCY] Davidson, A., Finkel, M., Thomson, M., and C. A. Wood, "Key Consistency and Discovery", Work in Progress, Internet-Draft, draft-ietf-privacypass-key-consistency-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-privacypass-key-consistency-01>.

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

[DMS2004] Dingledine, R., Mathewson, N., and P. Syverson, "Tor: The Second-Generation Onion Router", May 2004, <https://svn.torproject.org/svn/projects/design-paper/tor-design.html>.

[FIELDING] Fielding, R. T., "Architectural Styles and the Design of Network-based Software Architectures", January 2000, <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[FORWARDED] Petersson, A. and M. Nilsson, "Forwarded HTTP Extension", RFC 7239, DOI 10.17487/RFC7239, June 2014, <https://www.rfc-editor.org/info/rfc7239>.

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[NTP] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[ODOH] Kinnear, E., McManus, P., Pauly, T., Verma, T., and C.A. Wood, "Oblivious DNS over HTTPS", RFC 9230, DOI 10.17487/RFC9230, June 2022, <https://www.rfc-editor.org/info/rfc9230>.

[ODOH-PETS] Singanamalla, S., Chunhapanya, S., Hoyland, J., Vavruša, M., Verma, T., Wu, P., Fayed, M., Heimerl, K., Sullivan, N., and C. A. Wood, "Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS", PoPETS Proceedings Volume 2021, Issue 4, pp. 575-592, DOI 10.2478/popets-2021-0085, January 2021, <https://www.petsymposium.org/2021/files/papers/issue4/popets-2021-0085.pdf>.

[OHTTP-ANALYSIS] Hoyland, J., "Tamarin Model of Oblivious HTTP", commit 6824eee, October 2022, <https://github.com/cloudflare/ohttp-analysis>.

[PRIO] Corrigan-Gibbs, H. and D. Boneh, "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics", March 2017, <https://crypto.stanford.edu/prio/paper.pdf>.

[RANDOM] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[TEMPLATE] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[UWT] Nottingham, M., "Unsanctioned Web Tracking", W3C TAG Finding, July 2015, <https://www.w3.org/2001/tag/doc/unsanctioned-tracking/>.

[X25519] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

Appendix A. Complete Example of a Request and Response

A single request and response exchange is shown here. Binary values (key configuration, secret keys, the content of messages, and intermediate values) are shown in hexadecimal. The request and response here are minimal; the purpose of this example is to show the cryptographic operations. In this example, the Client is configured with the Oblivious Relay Resource URI of `https://proxy.example.org/request.example.net/proxy`, and the proxy is configured to map requests to this URI to the Oblivious Gateway Resource URI `https://example.com/oblivious/request`. The Target Resource URI, i.e., the resource the Client ultimately wishes to query, is `https://example.com`.

To begin the process, the Oblivious Gateway Resource generates a key pair. In this example, the server chooses DHKEM(X25519, HKDF-SHA256) and generates an X25519 key pair [X25519]. The X25519 secret key is:

```
3c168975674b2fa8e465970b79c8dcf09f1c741626480bd4c6162fc5b6a98e1a
```

The Oblivious Gateway Resource constructs a key configuration that includes the corresponding public key as follows:

```
01002031e1f05a740102115220e9af918f738674aec95f54db6e04eb705aae8e
79815500080001000100010003
```

This key configuration is somehow obtained by the Client. Then, when a Client wishes to send an HTTP GET request to the target `https://example.com`, it constructs the following binary HTTP message:

```
00034745540568747470730b6578616d706c652e636f6d012f
```

The Client then reads the Oblivious Gateway Resource key configuration and selects a mutually supported KDF and AEAD. In this example, the Client selects HKDF-SHA256 and AES-128-GCM. The Client then generates an HPKE sending context that uses the server public key. This context is constructed from the following ephemeral secret key:

```
bc51d5e930bda26589890ac7032f70ad12e4ecb37abb1b65b1256c9c48999c73
```

The corresponding public key is:

```
4b28f881333e7c164ffc499ad9796f877f4e1051ee6d31bad19dec96c208b472
```

The context is created with an `info` parameter of:

```
6d6573736167652f626874747020726571756573740001002000010001
```

Applying the Seal operation from the HPKE context produces an encrypted message, allowing the Client to construct the following Encapsulated Request:

```
010020000100014b28f881333e7c164ffc499ad9796f877f4e1051ee6d31bad1
9dec96c208b4726374e469135906992e1268c594d2a10c695d858c40a026e796
5e7d86b83dd440b2c0185204b4d63525
```

The Client then sends this to the Oblivious Relay Resource in a POST request, which might look like the following HTTP/1.1 request:

```
POST /request.example.net/proxy HTTP/1.1
Host: proxy.example.org
Content-Type: message/ohttp-req
Content-Length: 78

<content is the Encapsulated Request above>
```

The Oblivious Relay Resource receives this request and forwards it to the Oblivious Gateway Resource, which might look like:

```
POST /oblivious/request HTTP/1.1
Host: example.com
Content-Type: message/ohttp-req
Content-Length: 78

<content is the Encapsulated Request above>
```

The Oblivious Gateway Resource receives this request, selects the key it generated previously using the key identifier from the message, and decrypts the message. As this request is directed to the same server, the Oblivious Gateway Resource does not need to initiate an HTTP request to the Target Resource. The request can be served directly by the Target Resource, which generates a minimal response (consisting of just a 200 status code) as follows:

```
0140c8
```

The response is constructed by exporting a secret from the HPKE context:

```
62d87a6ba569ee81014c2641f52bea36
```

The key derivation for the Encapsulated Response uses both the encapsulated KEM key from the request and a randomly selected nonce. This produces a salt of:

```
4b28f881333e7c164ffc499ad9796f877f4e1051ee6d31bad19dec96c208b472
c789e7151fcba46158ca84b04464910d
```

The salt and secret are both passed to the `Extract` function of the selected KDF (HKDF-SHA256) to produce a pseudorandom key of:

```
979aaeae066cf211ab407b31ae49767f344e1501e475c84e8aff547cc5a683db
```

The pseudorandom key is used with the Expand function of the KDF and an info field of "key" to produce a 16-byte key for the selected AEAD (AES-128-GCM):

```
5d0172a080e428b16d298c4ea0db620d
```

With the same KDF and pseudorandom key, an info field of "nonce" is used to generate a 12-byte nonce:

```
f6bf1aeb88d6df87007fa263
```

The AEAD Seal() function is then used to encrypt the response, which is added to the randomized nonce value to produce the Encapsulated Response:

```
c789e7151fcba46158ca84b04464910d86f9013e404feea014e7be4a441f234f
857fbd
```

The Oblivious Gateway Resource constructs a response with the same content:

```
HTTP/1.1 200 OK
Date: Wed, 27 Jan 2021 04:45:07 GMT
Cache-Control: private, no-store
Content-Type: message/ohttp-res
Content-Length: 38

<content is the Encapsulated Response>
```

The same response might then be generated by the Oblivious Relay Resource, which might change as little as the Date header. The Client is then able to use the HPKE context it created and the nonce from the Encapsulated Response to construct the AEAD key and nonce and decrypt the response.

Acknowledgments

This design is based on a design for Oblivious DNS (queries) over HTTPS (DoH), described in [ODOH]. David Benjamin, Mark Nottingham, and Eric Rescorla made technical contributions. The authors also thank Ralph Giles, Lucas Pardue, and Tommy Pauly for invaluable assistance.

Authors' Addresses

Martin Thomson
Mozilla
Email: mt@lowentropy.net

Christopher A. Wood
Cloudflare
Email: caw@heapingbits.net