Stream:	Independent Submission
RFC:	9230
Category:	Experimental
Published:	June 2022
ISSN:	2070-1721

Authors:	E. Kinnear	P. McManus	T. Pauly	T. Verma	C.A. Wood
	Apple Inc.	*Fastly*	*Apple Inc.*	*Cloudflare*	*Cloudflare*

RFC 9230
Oblivious DNS over HTTPS

Abstract

This document describes a protocol that allows clients to hide their IP addresses from DNS resolvers via proxying encrypted DNS over HTTPS (DoH) messages. This improves privacy of DNS operations by not allowing any one server entity to be aware of both the client IP address and the content of DNS queries and answers.

This experimental protocol has been developed outside the IETF and is published here to guide implementation, ensure interoperability among implementations, and enable wide-scale experimentation.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9230.

Copyright Notice

Table of Contents

1. Introduction

DNS over HTTPS (DoH) [RFC8484] defines a mechanism to allow DNS messages to be transmitted in HTTP messages protected with TLS. This provides improved confidentiality and authentication for DNS interactions in various circumstances.

While DoH can prevent eavesdroppers from directly reading the contents of DNS exchanges, clients cannot send DNS queries to and receive answers from servers without revealing their local IP address (and thus information about the identity or location of the client) to the server.

Proposals such as Oblivious DNS [OBLIVIOUS-DNS] increase privacy by ensuring that no single DNS server is aware of both the client IP address and the message contents.

This document defines Oblivious DoH, an experimental protocol built on DoH that permits proxied resolution, in which DNS messages are encrypted so that no server can independently read both the client IP address and the DNS message contents.

As with DoH, DNS messages exchanged over Oblivious DoH are fully formed DNS messages. Clients that want to receive answers that are relevant to the network they are on without revealing their exact IP address can thus use the EDNS0 Client Subnet option ([RFC7871], Section 7.1.2) to provide a hint to the resolver using Oblivious DoH.

This mechanism is intended to be used as one mechanism for resolving privacy-sensitive content in the broader context of DNS privacy.

This experimental protocol has been developed outside the IETF and is published here to guide implementation, ensure interoperability among implementations, and enable wide-scale experimentation. See Section 10 for more details about the experiment.

1.1. Specification of Requirements

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.

2. Terminology

This document defines the following terms:

Oblivious Client: A client that sends DNS queries to an Oblivious Target, through an Oblivious Proxy. The Client is responsible for selecting the combination of Proxy and Target to use for a given query.

Oblivious Proxy: An HTTP server that proxies encrypted DNS queries and responses between an Oblivious Client and an Oblivious Target and is identified by a URI Template [RFC6570] (see Section 4.1). Note that this Oblivious Proxy is not acting as a full HTTP proxy but is instead a specialized server used to forward Oblivious DNS messages.

Oblivious Target: An HTTP server that receives and decrypts encrypted Oblivious Client DNS queries from an Oblivious Proxy and returns encrypted DNS responses via that same Proxy. In order to provide DNS responses, the Target can be a DNS resolver, be co-located with a resolver, or forward to a resolver.

Throughout the rest of this document, we use the terms "Client", "Proxy", and "Target" to refer to an Oblivious Client, Oblivious Proxy, and Oblivious Target, respectively.

3. Deployment Requirements

Oblivious DoH requires, at a minimum:

- An Oblivious Proxy server, identified by a URI Template.
- An Oblivious Target server. The Target and Proxy are expected to be non-colluding (see Section 11).
- One or more Target public keys for encrypting DNS queries sent to a Target via a Proxy (Section 5). These keys guarantee that only the intended Target can decrypt Client queries.

The mechanism for discovering and provisioning the Proxy URI Template and Target public keys is out of scope for this document.

4. HTTP Exchange

Unlike direct resolution, oblivious hostname resolution over DoH involves three parties:

1. The Client, which generates queries.
2. The Proxy, which receives encrypted queries from the Client and passes them on to a Target.
3. The Target, which receives proxied queries from the Client via the Proxy and produces proxied answers.

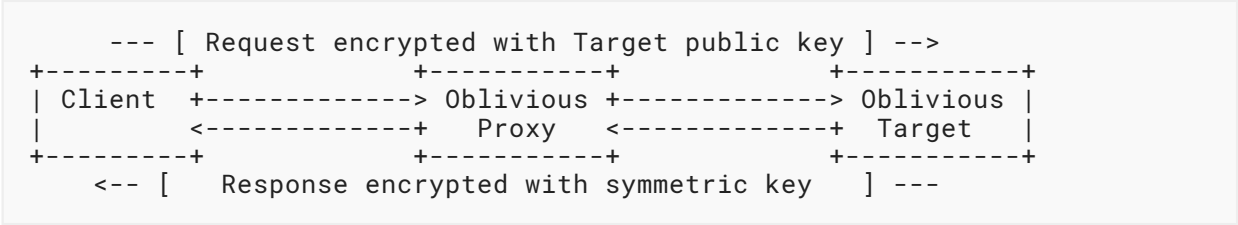

Figure 1: Oblivious DoH Exchange

4.1. HTTP Request

Oblivious DoH queries are created by the Client and are sent to the Proxy as HTTP requests using the POST method. Clients are configured with a Proxy URI Template [RFC6570] and the Target URI. The scheme for both the Proxy URI Template and the Target URI **MUST** be "https". The Proxy URI Template uses the Level 3 encoding defined in Section 1.2 of [RFC6570] and contains two variables: "targethost", which indicates the hostname of the Target server; and "targetpath", which indicates the path on which the Target is accessible. Examples of Proxy URI Templates are shown below:

```
https://dnsproxy.example/dns-query{?targethost,targetpath}
https://dnsproxy.example/{targethost}/{targetpath}
```

The URI Template **MUST** contain both the "targethost" and "targetpath" variables exactly once and **MUST NOT** contain any other variables. The variables **MUST** be within the path or query components of the URI. Clients **MUST** ignore configurations that do not conform to this template. See Section 4.2 for an example request.

Oblivious DoH messages have no cache value, since both requests and responses are encrypted using ephemeral key material. Requests and responses **MUST NOT** be cached.

Clients **MUST** set the HTTP Content-Type header to "application/oblivious-dns-message" to indicate that this request is an Oblivious DoH query intended for proxying. Clients also **SHOULD** set this same value for the HTTP Accept header.

A correctly encoded request has the HTTP Content-Type header "application/oblivious-dns-message", uses the HTTP POST method, and contains "targethost" and "targetpath" variables. If the Proxy fails to match the "targethost" and "targetpath" variables from the path, it **MUST** treat the request as malformed. The Proxy constructs the URI of the Target with the "https" scheme, using the value of "targethost" as the URI host and the percent-decoded value of "targetpath" as the URI path. Proxies **MUST** check that Client requests are correctly encoded and **MUST** return a 4xx (Client Error) if the check fails, along with the Proxy-Status response header with an "error" parameter of type "http_request_error" [RFC9209].

Proxies **MAY** choose to not forward connections to non-standard ports. In such cases, Proxies can indicate the error with a 403 response status code, along with a Proxy-Status response header with an "error" parameter of type "http_request_denied" and with an appropriate explanation in "details".

If the Proxy cannot establish a connection to the Target, it can indicate the error with a 502 response status code, along with a Proxy-Status response header with an "error" parameter whose type indicates the reason. For example, if DNS resolution fails, the error type might be "dns_timeout", whereas if the TLS connection fails, the error type might be "tls_protocol_error".

Upon receipt of requests from a Proxy, Targets **MUST** validate that the request has the HTTP Content-Type header "application/oblivious-dns-message" and uses the HTTP POST method. Targets can respond with a 4xx response status code if this check fails.

4.2. HTTP Request Example

The following example shows how a Client requests that a Proxy, "dnsproxy.example", forward an encrypted message to "dnstarget.example". The URI Template for the Proxy is "https://dnsproxy.example/dns-query{?targethost,targetpath}". The URI for the Target is "https://dnstarget.example/dns-query".

```
:method = POST
:scheme = https
:authority = dnsproxy.example
:path = /dns-query?targethost=dnstarget.example&targetpath=/dns-query
accept = application/oblivious-dns-message
content-type = application/oblivious-dns-message
content-length = 106

<Bytes containing an encrypted Oblivious DNS query>
```

The Proxy then sends the following request on to the Target:

```
:method = POST
:scheme = https
:authority = dnstarget.example
:path = /dns-query
accept = application/oblivious-dns-message
content-type = application/oblivious-dns-message
content-length = 106

<Bytes containing an encrypted Oblivious DNS query>
```

4.3. HTTP Response

The response to an Oblivious DoH query is generated by the Target. It **MUST** set the Content-Type HTTP header to "application/oblivious-dns-message" for all successful responses. The body of the response contains an encrypted DNS message; see Section 6.

The response from a Target **MUST** set the Content-Type HTTP header to "application/oblivious-dns-message", and that same type **MUST** be used on all successful responses sent by the Proxy to the Client. A Client **MUST** only consider a response that contains the Content-Type header before

processing the payload. A response without the appropriate header **MUST** be treated as an error and be handled appropriately. All other aspects of the HTTP response and error handling are inherited from standard DoH.

Proxies forward responses from the Target to the Client, without any modifications to the body or status code. The Proxy also **SHOULD** add a Proxy-Status response header with a "received-status" parameter indicating that the status code was generated by the Target.

Note that if a Client receives a 3xx status code and chooses to follow a redirect, the subsequent request **MUST** also be performed through a Proxy in order to avoid directly exposing requests to the Target.

Requests that cannot be processed by the Target result in 4xx (Client Error) responses. If the Target and Client keys do not match, it is an authorization failure (HTTP status code 401; see Section 15.5.2 of [HTTP]). Otherwise, if the Client's request is invalid, such as in the case of decryption failure, wrong message type, or deserialization failure, this is a bad request (HTTP status code 400; see Section 15.5.1 of [HTTP]).

Even in the case of DNS responses indicating failure, such as SERVFAIL or NXDOMAIN, a successful HTTP response with a 2xx status code is used as long as the DNS response is valid. This is identical to how DoH [RFC8484] handles HTTP response codes.

4.4. HTTP Response Example

The following example shows a 2xx (Successful) response that can be sent from a Target to a Client via a Proxy.

```
:status = 200
content-type = application/oblivious-dns-message
content-length = 154

<Bytes containing an encrypted Oblivious DNS response>
```

4.5. HTTP Metadata

Proxies forward requests and responses between Clients and Targets as specified in Section 4.1. Metadata sent with these messages could inadvertently weaken or remove Oblivious DoH privacy properties. Proxies **MUST NOT** send any Client-identifying information about Clients to Targets, such as "Forwarded" HTTP headers [RFC7239]. Additionally, Clients **MUST NOT** include any private state in requests to Proxies, such as HTTP cookies. See Section 11.3 for related discussion about Client authentication information.

5. Configuration and Public Key Format

In order to send a message to a Target, the Client needs to know a public key to use for encrypting its queries. The mechanism for discovering this configuration is out of scope for this document.

Servers ought to rotate public keys regularly. It is **RECOMMENDED** that servers rotate keys every day. Shorter rotation windows reduce the anonymity set of Clients that might use the public key, whereas longer rotation windows widen the time frame of possible compromise.

An Oblivious DNS public key configuration is a structure encoded, using TLS-style encoding [RFC8446], as follows:

```
struct {
   uint16 kem_id;
   uint16 kdf_id;
   uint16 aead_id;
   opaque public_key<1..2^16-1>;
} ObliviousDoHConfigContents;

struct {
   uint16 version;
   uint16 length;
   select (ObliviousDoHConfig.version) {
      case 0x0001: ObliviousDoHConfigContents contents;
   }
} ObliviousDoHConfig;

ObliviousDoHConfig ObliviousDoHConfigs<1..2^16-1>;
```

The `ObliviousDoHConfigs` structure contains one or more `ObliviousDoHConfig` structures in decreasing order of preference. This allows a server to support multiple versions of Oblivious DoH and multiple sets of Oblivious DoH parameters.

An `ObliviousDoHConfig` structure contains a versioned representation of an Oblivious DoH configuration, with the following fields.

version: The version of Oblivious DoH for which this configuration is used. Clients **MUST** ignore any `ObliviousDoHConfig` structure with a version they do not support. The version of Oblivious DoH specified in this document is `0x0001`.

length: The length, in bytes, of the next field.

contents: An opaque byte string whose contents depend on the version. For this specification, the contents are an `ObliviousDoHConfigContents` structure.

An `ObliviousDoHConfigContents` structure contains the information needed to encrypt a message under `ObliviousDoHConfigContents.public_key` such that only the owner of the corresponding private key can decrypt the message. The values for `ObliviousDoHConfigContents.kem_id`, `ObliviousDoHConfigContents.kdf_id`, and `ObliviousDoHConfigContents.aead_id` are described in Section 7 of [HPKE]. The fields in this structure are as follows:

kem_id:

The hybrid public key encryption (HPKE) key encapsulation mechanism (KEM) identifier corresponding to `public_key`. Clients **MUST** ignore any `ObliviousDoHConfig` structure with a key using a KEM they do not support.

kdf_id: The HPKE key derivation function (KDF) identifier corresponding to `public_key`. Clients **MUST** ignore any `ObliviousDoHConfig` structure with a key using a KDF they do not support.

aead_id: The HPKE authenticated encryption with associated data (AEAD) identifier corresponding to `public_key`. Clients **MUST** ignore any `ObliviousDoHConfig` structure with a key using an AEAD they do not support.

public_key: The HPKE public key used by the Client to encrypt Oblivious DoH queries.

6. Protocol Encoding

This section includes encoding and wire format details for Oblivious DoH, as well as routines for encrypting and decrypting encoded values.

6.1. Message Format

There are two types of Oblivious DoH messages: Queries (0x01) and Responses (0x02). Both messages carry the following information:

1. A DNS message, which is either a Query or Response, depending on context.
2. Padding of arbitrary length, which **MUST** contain all zeros.

They are encoded using the following structure:

```
struct {
   opaque dns_message<1..2^16-1>;
   opaque padding<0..2^16-1>;
} ObliviousDoHMessagePlaintext;
```

Both Query and Response messages use the `ObliviousDoHMessagePlaintext` format.

```
ObliviousDoHMessagePlaintext ObliviousDoHQuery;
ObliviousDoHMessagePlaintext ObliviousDoHResponse;
```

An encrypted `ObliviousDoHMessagePlaintext` parameter is carried in an `ObliviousDoHMessage` message, encoded as follows:

```
struct {
   uint8  message_type;
   opaque key_id<0..2^16-1>;
   opaque encrypted_message<1..2^16-1>;
} ObliviousDoHMessage;
```

The `ObliviousDoHMessage` structure contains the following fields:

message_type: A one-byte identifier for the type of message. Query messages use `message_type` 0x01, and Response messages use `message_type` 0x02.

key_id: The identifier of the corresponding `ObliviousDoHConfigContents` key. This is computed as `Expand(Extract("", config), "odoh key id", Nh)`, where `config` is the `ObliviousDoHConfigContents` structure and `Extract`, `Expand`, and `Nh` are as specified by the HPKE cipher suite KDF corresponding to `config.kdf_id`.

encrypted_message: An encrypted message for the Oblivious Target (for Query messages) or Client (for Response messages). Implementations **MAY** enforce limits on the size of this field, depending on the size of plaintext DNS messages. (DNS queries, for example, will not reach the size limit of 2^16-1 in practice.)

The contents of `ObliviousDoHMessage.encrypted_message` depend on `ObliviousDoHMessage.message_type`. In particular, `ObliviousDoHMessage.encrypted_message` is an encryption of an `ObliviousDoHQuery` message if the message is a Query and an encryption of `ObliviousDoHResponse` if the message is a Response.

6.2. Encryption and Decryption Routines

Clients use the following utility functions for encrypting a Query and decrypting a Response as described in Section 7.

- encrypt_query_body: Encrypt an Oblivious DoH query.

```
def encrypt_query_body(pkR, key_id, Q_plain):
  enc, context = SetupBaseS(pkR, "odoh query")
  aad = 0x01 || len(key_id) || key_id
  ct = context.Seal(aad, Q_plain)
  Q_encrypted = enc || ct
  return Q_encrypted
```

- decrypt_response_body: Decrypt an Oblivious DoH response.

```
def decrypt_response_body(context, Q_plain, R_encrypted, resp_nonce):
  aead_key, aead_nonce = derive_secrets(context, Q_plain, resp_nonce)
  aad = 0x02 || len(resp_nonce) || resp_nonce
  R_plain, error = Open(key, nonce, aad, R_encrypted)
  return R_plain, error
```

The `derive_secrets` function is described below.

Targets use the following utility functions in processing queries and producing responses as described in Section 8.

- setup_query_context: Set up an HPKE context used for decrypting an Oblivious DoH query.

```
def setup_query_context(skR, key_id, Q_encrypted):
  enc || ct = Q_encrypted
  context = SetupBaseR(enc, skR, "odoh query")
  return context
```

- decrypt_query_body: Decrypt an Oblivious DoH query.

```
def decrypt_query_body(context, key_id, Q_encrypted):
  aad = 0x01 || len(key_id) || key_id
  enc || ct = Q_encrypted
  Q_plain, error = context.Open(aad, ct)
  return Q_plain, error
```

- derive_secrets: Derive keying material used for encrypting an Oblivious DoH response.

```
def derive_secrets(context, Q_plain, resp_nonce):
  secret = context.Export("odoh response", Nk)
  salt = Q_plain || len(resp_nonce) || resp_nonce
  prk = Extract(salt, secret)
  key = Expand(odoh_prk, "odoh key", Nk)
  nonce = Expand(odoh_prk, "odoh nonce", Nn)
  return key, nonce
```

The `random(N)` function returns `N` cryptographically secure random bytes from a good source of entropy [RFC4086]. The `max(A, B)` function returns `A` if `A > B`, and `B` otherwise.

- encrypt_response_body: Encrypt an Oblivious DoH response.

```
def encrypt_response_body(R_plain, aead_key, aead_nonce, resp_nonce):
  aad = 0x02 || len(resp_nonce) || resp_nonce
  R_encrypted = Seal(aead_key, aead_nonce, aad, R_plain)
  return R_encrypted
```

7. Oblivious Client Behavior

Let `M` be a DNS message (query) a Client wishes to protect with Oblivious DoH. When sending an Oblivious DoH Query for resolving `M` to an Oblivious Target with `ObliviousDoHConfigContents config`, a Client does the following:

1. Creates an `ObliviousDoHQuery` structure, carrying the message M and padding, to produce Q_plain.
2. Deserializes `config.public_key` to produce a public key pkR of type `config.kem_id`.
3. Computes the encrypted message as `Q_encrypted = encrypt_query_body(pkR, key_id, Q_plain)`, where `key_id` is as computed in Section 6. Note also that `len(key_id)` outputs the length of `key_id` as a two-byte unsigned integer.
4. Outputs an `ObliviousDoHMessage` message `Q`, where `Q.message_type = 0x01`, `Q.key_id` carries `key_id`, and `Q.encrypted_message = Q_encrypted`.

The Client then sends `Q` to the Proxy according to Section 4.1. Once the Client receives a response `R`, encrypted as specified in Section 8, it uses `decrypt_response_body` to decrypt `R.encrypted_message` (using `R.key_id` as a nonce) and produce R_plain. Clients **MUST** validate `R_plain.padding` (as all zeros) before using `R_plain.dns_message`.

8. Oblivious Target Behavior

Targets that receive a Query message Q decrypt and process it as follows:

1. Look up the `ObliviousDoHConfigContents` information according to `Q.key_id`. If no such key exists, the Target **MAY** discard the query, and if so, it **MUST** return a 401 (Unauthorized) response to the Proxy. Otherwise, let `skR` be the private key corresponding to this public key, or one chosen for trial decryption.
2. Compute `context = setup_query_context(skR, Q.key_id, Q.encrypted_message)`.
3. Compute `Q_plain, error = decrypt_query_body(context, Q.key_id, Q.encrypted_message)`.
4. If no error was returned and `Q_plain.padding` is valid (all zeros), resolve `Q_plain.dns_message` as needed, yielding a DNS message M. Otherwise, if an error was returned or the padding was invalid, return a 400 (Client Error) response to the Proxy.
5. Create an `ObliviousDoHResponseBody` structure, carrying the message `M` and padding, to produce `R_plain`.
6. Create a fresh nonce `resp_nonce = random(max(Nn, Nk))`.
7. Compute `aead_key, aead_nonce = derive_secrets(context, Q_plain, resp_nonce)`.
8. Compute `R_encrypted = encrypt_response_body(R_plain, aead_key, aead_nonce, resp_nonce)`. The `key_id` field used for encryption carries `resp_nonce` in order for Clients to derive the same secrets. Also, the `Seal` function is the function that is associated with the HPKE AEAD.

9. Output an `ObliviousDoHMessage` message `R`, where `R.message_type = 0x02`, `R.key_id = resp_nonce`, and `R.encrypted_message = R_encrypted`.

The Target then sends `R` in a 2xx (Successful) response to the Proxy; see Section 4.3. The Proxy forwards the message `R` without modification back to the Client as the HTTP response to the Client's original HTTP request. In the event of an error (non-2xx status code), the Proxy forwards the Target error to the Client; see Section 4.3.

9. Compliance Requirements

Oblivious DoH uses HPKE for public key encryption [HPKE]. In the absence of an application profile standard specifying otherwise, a compliant Oblivious DoH implementation **MUST** support the following HPKE cipher suite:

KEM: DHKEM(X25519, HKDF-SHA256) (see [HPKE], Section 7.1)

KDF: HKDF-SHA256 (see [HPKE], Section 7.2)

AEAD: AES-128-GCM (see [HPKE], Section 7.3)

10. Experiment Overview

This document describes an experimental protocol built on DoH. The purpose of this experiment is to assess deployment configuration viability and related performance impacts on DNS resolution by measuring key performance indicators such as resolution latency. Experiment participants will test various parameters affecting service operation and performance, including mechanisms for discovery and configuration of DoH Proxies and Targets, as well as performance implications of connection reuse and pools where appropriate. The results of this experiment will be used to influence future protocol design and deployment efforts related to Oblivious DoH, such as Oblivious HTTP [OHTP]. Implementations of DoH that are not involved in the experiment will not recognize this protocol and will not participate in the experiment. It is anticipated that the use of Oblivious DoH will be widespread and that this experiment will be of long duration.

11. Security Considerations

Oblivious DoH aims to keep knowledge of the true query origin and its contents known only to Clients. As a simplified model, consider a case where there exist two Clients C1 and C2, one Proxy P, and one Target T. Oblivious DoH assumes an extended Dolev-Yao style attacker [Dolev-Yao] that can observe all network activity and can adaptively compromise either P or T, but not C1 or C2. Note that compromising both P and T is equivalent to collusion between these two parties in practice. Once compromised, the attacker has access to all session information and private key material. (This generalizes to arbitrarily many Clients, Proxies, and Targets, with the constraints that (1) not all Targets and Proxies are simultaneously compromised and (2) at least two Clients are left uncompromised.) The attacker is prohibited from sending Client-identifying information, such as IP addresses, to Targets. (This would allow the attacker to trivially link a query to the corresponding Client.)

In this model, both C1 and C2 send Oblivious DoH queries Q1 and Q2, respectively, through P to T, and T provides answers A1 and A2. The attacker aims to link C1 to (Q1, A1) and C2 to (Q2, A2), respectively. The attacker succeeds if this linkability is possible without any additional interaction. (For example, if T is compromised, it could return a DNS answer corresponding to an entity it controls and then observe the subsequent connection from a Client, learning its identity in the process. Such attacks are out of scope for this model.)

Oblivious DoH security prevents such linkability. Informally, this means:

1. Queries and answers are known only to Clients and Targets in possession of the corresponding response key and HPKE keying material. In particular, Proxies know the origin and destination of an oblivious query, yet do not know the plaintext query. Likewise, Targets know only the oblivious query origin, i.e., the Proxy, and the plaintext query. Only the Client knows both the plaintext query contents and destination.
2. Target resolvers cannot link queries from the same Client in the absence of unique per-Client keys.

Traffic analysis mitigations are outside the scope of this document. In particular, this document does not prescribe padding lengths for `ObliviousDoHQuery` and `ObliviousDoHResponse` messages. Implementations **SHOULD** follow the guidance in [RFC8467] for choosing padding length.

Oblivious DoH security does not depend on Proxy and Target indistinguishability. Specifically, an on-path attacker could determine whether a connection to a specific endpoint is used for oblivious or direct DoH queries. However, this has no effect on the confidentiality goals listed above.

11.1. Denial of Service

Malicious Clients (or Proxies) can send bogus Oblivious DoH queries to Targets as a Denial-of-Service (DoS) attack. Target servers can throttle processing requests if such an event occurs. Additionally, since Targets provide explicit errors upon decryption failure, i.e., if ciphertext decryption fails or if the plaintext DNS message is malformed, Proxies can throttle specific Clients in response to these errors. In general, however, Targets trust Proxies to not overwhelm the Target, and it is expected that Proxies implement either some form of rate limiting or client authentication to limit abuse; see Section 11.3.

Malicious Targets or Proxies can send bogus answers in response to Oblivious DoH queries. Response decryption failure is a signal that either the Proxy or Target is misbehaving. Clients can choose to stop using one or both of these servers in the event of such failure. However, as noted above, malicious Targets and Proxies are out of scope for the threat model.

11.2. Proxy Policies

Proxies are free to enforce any forwarding policy they desire for Clients. For example, they can choose to only forward requests to known or otherwise trusted Targets.

Proxies that do not reuse connections to Targets for many Clients may allow Targets to link individual queries to unknown Targets. To mitigate this linkability vector, it is **RECOMMENDED** that Proxies pool and reuse connections to Targets. Note that this benefits performance as well as privacy, since queries do not incur any delay that might otherwise result from Proxy-to-Target connection establishment.

11.3. Authentication

Depending on the deployment scenario, Proxies and Targets might require authentication before use. Regardless of the authentication mechanism in place, Proxies **MUST NOT** reveal any Client authentication information to Targets. This is required so Targets cannot uniquely identify individual Clients.

Note that if Targets require Proxies to authenticate at the HTTP or application layer before use, this ought to be done before attempting to forward any Client query to the Target. This will allow Proxies to distinguish 401 (Unauthorized) response codes due to authentication failure from 401 response codes due to Client key mismatch; see Section 4.3.

12. IANA Considerations

This document makes changes to the "Media Types" registry. The changes are described in the following subsection.

12.1. Oblivious DoH Message Media Type

This document registers a new media type, "application/oblivious-dns-message".

Type name: application

Subtype name: oblivious-dns-message

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: This is a binary format, containing encrypted DNS requests and responses encoded as `ObliviousDoHMessage` values, as defined in Section 6.1.

Security considerations: See this document. The content is an encrypted DNS message, and not executable code.

Interoperability considerations: This document specifies the format of conforming messages and the interpretation thereof; see Section 6.1.

Published specification: This document

Applications that use this media type: This media type is intended to be used by Clients wishing to hide their DNS queries when using DNS over HTTPS.

Additional information: N/A

Person and email address to contact for further information: See the Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: Tommy Pauly (tpauly@apple.com)

Change controller: IETF

Provisional registration? (standards tree only): No

13. References

13.1. Normative References

[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>.

[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>.

[RFC4086] 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>.

[RFC6570] 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>.

[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>.

[RFC8446] 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>.

[RFC8467] Mayrhofer, A., "Padding Policies for Extension Mechanisms for DNS (EDNS(0))", RFC 8467, DOI 10.17487/RFC8467, October 2018, <https://www.rfc-editor.org/info/rfc8467>.

[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>.

[RFC9209] Nottingham, M. and P. Sikora, "The Proxy-Status HTTP Response Header Field", RFC 9209, DOI 10.17487/RFC9209, June 2022, <https://www.rfc-editor.org/info/rfc9209>.

13.2. Informative References

[Dolev-Yao] Dolev, D. and A. C. Yao, "On the Security of Public Key Protocols", IEEE Transactions on Information Theory, Vol. IT-29, No. 2, DOI 10.1109/TIT.1983.1056650, March 1983, <https://www.cs.huji.ac.il/~dolev/pubs/dolev-yao-ieee-01056650.pdf>.

[OBLIVIOUS-DNS] Edmundson, A., Schmitt, P., Feamster, N., and A. Mankin, "Oblivious DNS - Strong Privacy for DNS Queries", Work in Progress, Internet-Draft, draft-annee-dprive-oblivious-dns-00, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-annee-dprive-oblivious-dns-00>.

[OHTP] Thomson, M. and C.A. Wood, "Oblivious HTTP", Work in Progress, Internet-Draft, draft-ietf-ohai-ohttp-01, 15 February 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-ohai-ohttp-01>.

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

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/info/rfc7871>.

Appendix A. Use of Generic Proxy Services

Using DoH over anonymizing proxy services such as Tor can also achieve the desired goal of separating query origins from their contents. However, there are several reasons why such systems are undesirable as contrasted with Oblivious DoH:

1. Tor is meant to be a generic connection-level anonymity system, and it incurs higher latency costs and protocol complexity for the purpose of proxying individual DNS queries. In contrast, Oblivious DoH is a lightweight protocol built on DoH, implemented as an application-layer proxy, that can be enabled as a default mode for users that need increased privacy.
2. As a one-hop proxy, Oblivious DoH encourages connectionless proxies to mitigate Client query correlation with few round trips. In contrast, multi-hop systems such as Tor often run secure connections (TLS) end to end, which means that DoH servers could track queries over the same connection. Using a fresh DoH connection per query would incur a non-negligible penalty in connection setup time.

Acknowledgments

This work is inspired by Oblivious DNS [OBLIVIOUS-DNS]. Thanks to all of the authors of that document. Thanks to Nafeez Ahamed, Elliot Briggs, Marwan Fayed, Jonathan Hoyland, Frederic Jacobs, Tommy Jensen, Erik Nygren, Paul Schmitt, Brian Swander, and Peter Wu for their feedback and input.

Authors' Addresses

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America
Email: ekinnear@apple.com

Patrick McManus
Fastly
Email: mcmanus@ducksong.com

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America
Email: tpauly@apple.com

Tanya Verma
Cloudflare
101 Townsend St
San Francisco, California 94107
United States of America
Email: vermatanyax@gmail.com

Christopher A. Wood
Cloudflare
101 Townsend St
San Francisco, California 94107
United States of America
Email: caw@heapingbits.net