Oblivious HTTP

draft-thomson-http-oblivious

SECDISPATCH, IETF 110, notPrague

*Martin Thomson, Chris Wood

What?

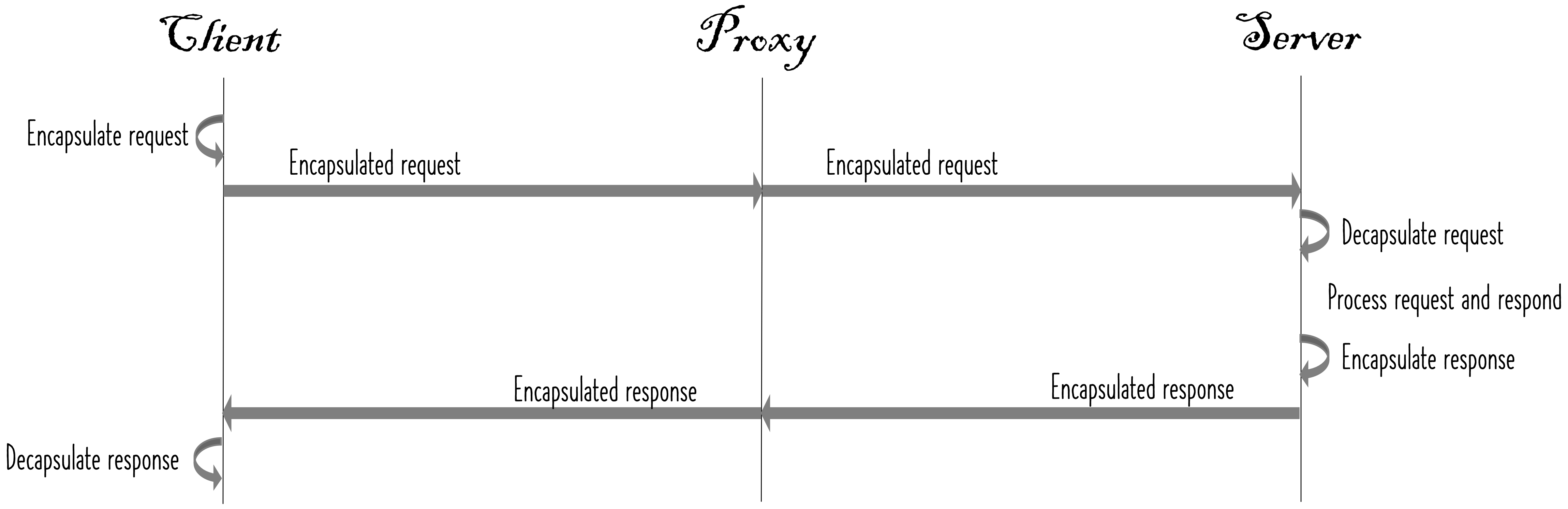
A system and method for making **unlinkable** HTTP requests

Comprising

- A proxy to hide source addressing and mix requests for traffic analysis resistance
- An additional layer of encryption to hide information from the proxy

How?

Server publishes its HPKE configuration; a fresh HPKE context is used for every exchange



Why?

- Clients might not want a server to link requests
- Examples
  - DNS queries to a resolver (see oblivious DNS)
  - Telemetry queries
- Less overhead than alternatives
  - A regular HTTP proxy with a connection per request has a lot of overhead
  - Tor has much stronger requirements, and much higher overheads
  - Prio is great for counting sensitive data, but adds delays and requires more infrastructure

Why not?

Not reasons not to standardize, just reasons not to use this always

- It is no good for general purpose HTTP (no state can carry between requests)
- It is more expensive than a direct request
- It isn't good enough where there is less trust (use something better suited)

Compared to one request per connection

Oblivious HTTP trades proxy replay protection, PCS security, and protocol changes for performance

A TLS connection for each request involves

- 1 ECDH keygen, 1 ECDH multiplication, 1 ECDSA signing or verification, lots of hashing
- 2 round trips (minimum) and lots of extra bytes

Oblivious HTTP involves

- 1 ECDH keygen (client only), 1 ECDH multiplication, a little less hashing
- 1 round trip and extra bytes (minimum 55, 32 for requests, responses plus HTTP wrapping)

Conditions

The proxy has limited trust from both client and server:

- The client trusts the proxy not to leak their identity to the server
- The server trusts the proxy not to overload it

Clients and servers might need to pad to resist traffic analysis

Servers need to protect against replay attacks from the proxy

Server compromise allows reading of messages if the proxy colludes

HTTP message format

- This could work with message/http
  - That is very difficult to implement correctly
  - Lots of security vulnerabilities there
- draft-thomson-http-binary-message is a simplified binary encoding based on HTTP/3
  - No header compression
  - Only flexibility is to allow streaming processing

Where?

Specification is small and largely self-contained

Interoperable implementations in Go and Rust (with test client and server)

- https://github.com/chris-wood/ohttp-go
- https://github.com/martinthomson/ohttp

Is there interest in doing the work?

Where should this be done?

- Suggest a short-lived working group (protocol only; defer discovery mechanisms)