# Building onion routing from CoRE building blocks

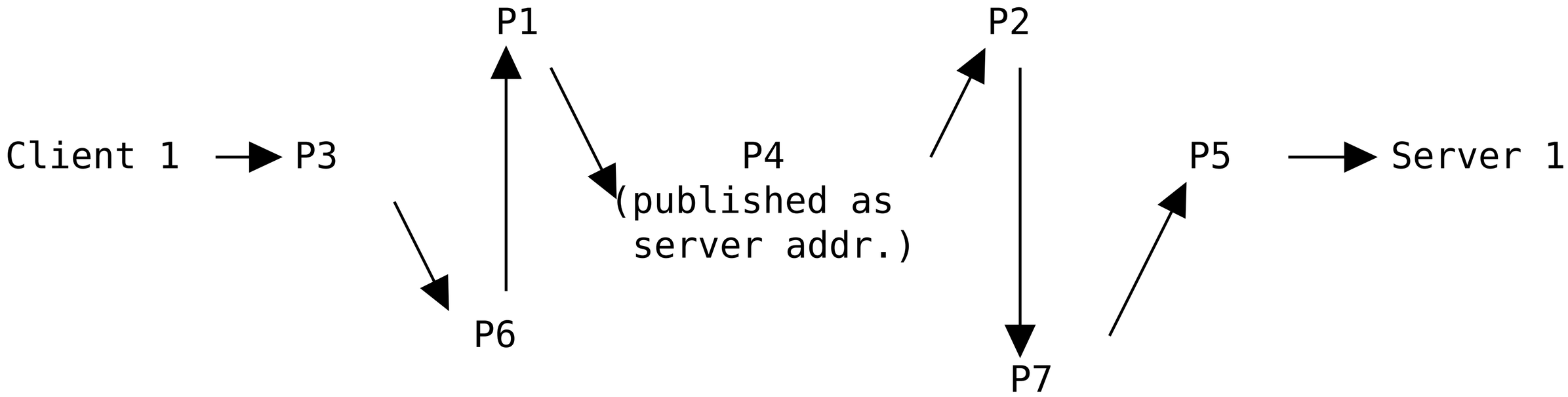
draft-amsuess-t2trg-onion-coap-02

*Christian Amsüss*, Marco Tiloca, Rikard Höglund

IETF121 Dublin, CoRE, 2024-11-07

# What this is about

## A request's path through an onion style network




Proxies selected randomly by server (P5, P7, P2, P4) and client (P3, P6, P1)

# CoRE building blocks this uses

$\alpha$ early, $\beta$ mature, $\square$ published, $*$ just for this, $?$ unsure classification

- $\square$ EDHOC and OSCORE
- $\beta$ OSCORE in OSCORE (`ietf-core-oscore-capable-proxies`), including its source routing for requests
- $\beta$ Advertising proxies (`ietf-core-transport-indication`)
- $\alpha$ Naming by hashes (`ietf-core-transport-indication`) ($*$ deriving names from EDHOC identities)
- $\beta^?$ Resource Directory (RD) doubling as proxy (`amsuess-core-resource-directory-extensions`) with some extras (optimizations?) in this document
- $*^?$ Globally federated address database (and how to tell RD to register there)

# OSCORE in OSCORE $\beta$

## Source routing from the client

```
POST to coap://proxy1
OSCORE: Encrypted with context client-proxy1          \
{ POST                                                | repeated
  Proxy-Scheme: coap                                  | layers
  Uri-Host: proxy2                                    /
  OSCORE: Encrypted with context client-proxy2        \
  { POST                                              |
    Proxy-Scheme: coap                                |
    Uri-Host: server                                  /
    OSCORE: Encrypted with context client-server
    { GET
      Uri-Path: /.well-known/core
} } }
```

# OSCORE in OSCORE $\beta$: Setup through EDHOC ?

## Step by step establishment

```
POST to coap://proxy1
OSCORE: Encrypted with context client-proxy1          \
{ POST                                                | n
  Proxy-Scheme: coap                                  | layers
  Uri-Host: proxy2                                    /
  OSCORE: Encrypted with context client-proxy2
  { POST
    Uri-Path: /.well-known/edhoc
    Payload:
    EDHOC message 1
    (establishing (n+1)th level of nesting)

   } }
```

# Naming things with hashes $\alpha$

`ietf-core-transport-indication`

`coap://edhoc-cred.uehks...3xjdqa.service.arpa`

"You will recognize the server by this KCCS[1]."

Other use cases: Expressing SVCB in literals; providing EDHOC authentication criteria in DNS-over-CoAP discovery.

[1] EDHOC/COSE lingo for raw public key

# Advertising proxies through hyperlinks $\beta$

`ietf-core-transport-indication`

```
<coap://[2001:db8::1]>;rel=has-proxy;\
anchor="coap://edhoc-cred.uehks…3xjdqa.service.arpa"
```

(a link from `coap://edh…arpa` with the relation `has-proxy` to `coap://[…]`).

"In order to reach that the `uehks…` URI, you can go through `2001:db8::1`."

Other use cases: providing pre-resolved names; discovering CoAP transports.

# Resource Directory for storing statements □

## RFC 9176

- POST some links to the resource directory
- Others can find those links.

Example links:

```
POST to coap://rd-host
Uri-Path: register
Payload:
</hello>
```

RD infers the endpoint's address as base address

# Adding in RD extensions $\beta^?$

`amsuess-core-resource-directory-extensions`, `ietf-core-groupcomm-proxy`

"Here are my links, and please proxy for me"

...but what if I am running virtual hosting? Role reversal address[2] is only an IP literal.

Suspecting to overlap with `Reply-From` option: "When interacting with me as a server, call me this in `Uri-Host`"

Is this strictly necessary or merely an optimization?

Is the role reversal address of an EDHOC established connection just the IP, or maybe constructed into `coap://edhoc-cred.uehks...3xjdqa.service.arpa`?

[2] Implied in existing RFCs, but lacking definition.

# Reverse proxying without hop-by-hop RD registration $*^{?}$

Proposed option: “Be prepared to send role reversal my way.”

Sets up reverse proxy ad hoc; relevant only when performing actions that cause role reversal.

Hop-by-hop, but set on each hop.

Similar semantics in ICNRG (“reflexive forwarding”).

Proxy may need to use `Reply-From` for multiplexing.

# Full sequence: Registering a hidden service

## S-P1-P2-P3; P3 is member of a federated RD

1. Establish EDHOC with P1
2. Using that as a forward proxy, establish EDHOC with P2
3. Using that as a forward proxy, establish EDHOC with P3
4. Through that, register at P3 RD.
    1. Allow-Role-Reverse inner option to P1
    2. P1 sets Allow-Role-Reverse option to P2, may need `Reply-From` for multiplexing; P2-P3 likewise
    3. P3 sees inner registration request.
    4. P3 sees “proxy for me”, and adds link to itself as proxy for the registered hidden service name.
5. Clients find cryptographic name in federated RD, and P3 as proxy address.

Questions?

Suggestions?