Cacheable OSCORE

Or "What to do when numeric request-response binding fails us".

`draft-amsuess-core-cachable-oscore-03`

Christian Amsüss, Marco Tiloca

2021-11-08, IETF 112

Development since IETF110: It's really two topics

I. How is request-response binding provided –
when the server does not get source authentication?

II. Once we know that, what do we need for cacheability?

Split introduced late in -03 – not as big as feared, but …directions?

Request-response binding in OSCORE

What would need to go wrong for response mismatch[1] to happen?

Client intends (and sends) $R1$.
Server processes (and answers to) $R2$.
OSCORE ensures sender and seqno match between $R1$ and $R2$.

Only client and server can produce such messages, and can thus trust them to be identical.

[1] See draft-mattsson-core-coap-attacks-01: CoAP Attacks

Request-response binding in Group OSCORE

What would need to go wrong for response mismatch to happen?

Client C intends (and sends) $R1$.
Server S processes (and answers to) $R2$.
OSCORE ensures sender and seqno match[2] between $R1$ and $R2$.

Only C and S can produce such messages *because of source authentication in all messages*.

[2] …and KID context, but that doesn't matter much here

Who can use a response?

In group/group mode, every member can read responses.

A third party T can only trust a captured[3] response when the original client *and* the server: Client C could have sent distinct $R1$ to be seen by T, and $R2$ to be seen by S.

[3]Or cached, we'll come to that

How can a response be made usable without trusting C?

- Full request is part of response
  e. g. a Class E or Class I Response-For[4]
- Hash of request is part of response (Class I or E)
- Either is part of the AAD without being part of the message at all
  e. g. by a "hidden Class I option" (currently in cacheable), or by extension of external_aad

...replacing / augmenting the (otherwise very practical) request-response binding mechanism.

[4] draft-bormann-core-responses-00: Non-traditional response forms

...and thus, Cacheable OSCORE is split

I Request-Response binding can be thusly managed – with some caveats described for Cacheable OSCORE (no freshness)

II Deterministic requests become a simple means to create common cache keys, and only deal with avoiding nonce reuse and limited request privacy

Questions

- Where else is part I useful?
- Is this simpler to follow when presented in split form inside a single document?

Answers? Other questions? Comments?