Observe Notifications as CoAP Multicast Responses

draft-tiloca-core-observe-multicast-notifications-03

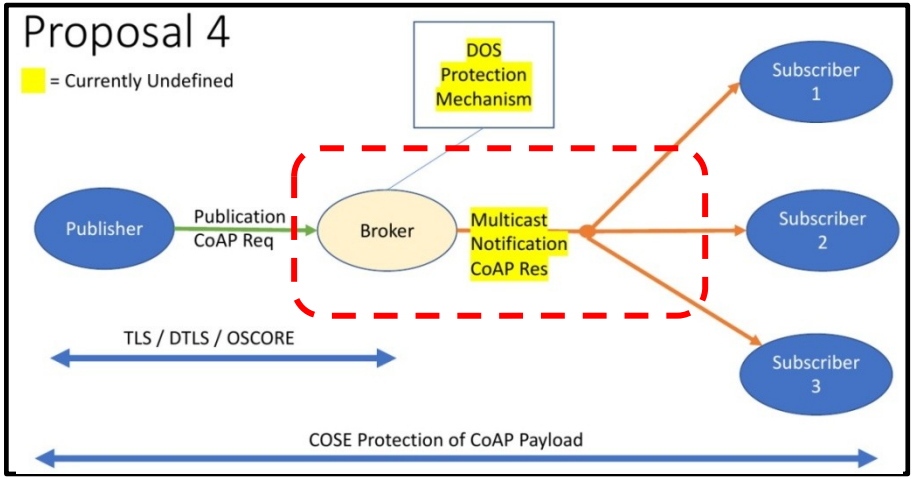
Marco Tiloca, RISE
Rikard Höglund, RISE
Christian Amsüss
Francesca Palombini, Ericsson

IETF 108, CoRE WG, July 31st, 2020

Recap

- Observe notifications as multicast responses
  - Many clients observe the same resource on a server S
  - Improved performance due to multicast delivery
  - Multicast responses are not defined yet. Token binding? Security?

- Example use case
  - Pub-Sub scenario
  - Many clients subscribe to a same topic on the Broker
  - Better performance
  - Subscribers are clients only



From the Hallway Discussion @ IETF 104

Proposed approach

- Define Observe notifications as multicast responses

- Token space from a group to a particular server
  - The Token space belongs to the group (clients)
  - The group entrusts the management to the server
  - All clients in a group observation use the same Token value

- Group OSCORE to protect multicast notifications
  - The server aligns all clients of an observation on a same *external_aad*
  - All notifications for a resource are protected with that *external_aad*

Phantom request and error response

- The server can start a group observation for a resource, e.g. :
  1. With no observers yet, a traditional registration request comes from a first client
  2. With many traditional observations, all clients are shifted to a group observation

- Consensus on token / external_aad by creating a Phantom observation request
  - Generated inside the server, it does not hit the wire
  - Like if sent by the group, from the multicast IP address of the group
  - Multicast notifications are responses to this phantom request

- To the unicast request, the server sends a 5.03 ***error response*** with:
  - Serialization of the phantom request
  - IP multicast address where notifications are sent to
  - Serialization of the latest multicast notification (i.e. current resource status)

Updates overview

Informative error response

```
Payload: { ph_req      : bstr(PH_REQ.CoAP),
           last_notif  : bstr(LAST_NOTIF.CoAP),
           cl_addr     : bstr(GROUP_ADDR),
           cl_port     : GROUP_PORT,
           srv_addr    : bstr(SERVER_ADDR),
           srv_port    : SERVER_PORT,
         }
```

- Revised encoding of the error response
- Parameter meaning
  - *ph_req* : serialization of the phantom request
  - *last_notif* : serialization of the latest sent multicast notification
  - *cl_addr* , *cl_port*: source address/port of the phantom request
    → Destination address/port of the multicast notifications
  - *srv_addr* , *srv_port*: destination address/port of the phantom request
- *'last_notif'* gives clients:
  - The current representation of the target resource
  - A baseline for the Observe number of following multicast notifications
  - May become optional – opinions?
- When creating the observation, the server creates and stores a first *'last_notif'*

Updates overview

- Improved rough counting of active clients
  - Poll for interest, using a new CoAP option in successful multicast notifications

- Server current rough estimate: *N*
  - Expected confirmations *M* < *N*
  - Option value: *Q* = ceil (*N* / *M*)
  - Each client picks a random *I* : [0, *Q*)
  - If *I* == 0, the client sends a re-registration request
    - Non Confirmable; w/ No-Response; w/ the new Option having empty value
    - Given explicit indications to prevent Smurf attacks
  - The server receives *R* of such requests; *X* new clients have registered in the meanwhile
    - Added a server timeout, building on RFC 7252 and *core-groupcomm-bis* parameters
  - Then *N* := (*R* * Q) + *X*

No.	C	U	N	R	Name	Format	Len.	Default
TBD		x			Multicast-Response-Feedback-Divider	uint	0-8 B	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,

- The new Appendix A describes the algorithm in pseudo-code

Updates overview

- Alternative ways to retrieve a phantom request
  - Revised examples in Appendix B
  - Pub-Sub (phantom request as part of topic metadata)
  - Sender introspection of intercepted notifications
- Congestion control
  - Added text about broadcast storm
- Clarifications on Group OSCORE
  - The group mode is the one to use

```
Request:

    GET </ps/topics?rt=oic.r.temperature>
    Accept: CoRAL

Response:

    2.05 Content
    Content-Format: CoRAL

    rdf:type <http://example.org/pubsub/topic-list>
    topic </ps/topics/1234> {
        ph_req h"120100006464b431323334"
        last_notif h"120100006464b431324321"
        cli_addr h"ff35003020010db8..1234"
        cli_port 5683
        srv_addr h"20010db80100..0001"
        srv_port 5683
    }
```

```
Request:

    GET </.well-known/core/mc-sender?token=6464>

Response:

    2.05 Content
    Content-Format: application/informative-response+cbor

    {
        'ph_req': h"120100006464b431323334"
        'last_notif' : h"120100006464b431324321"
        'cli_addr': h"ff35003020010db8..1234"
        'cli_port': 5683
        'srv_addr': h"20010db80100..0001"
        'srv_port': 5683
    }
```

Summary

- Multicast notifications to all clients observing a resource

- Latest additions
  - Improved encoding of error response
  - Improved rough counting of clients
  - Clarifications and editorial revision

- Next steps
  - Cover a scenario where a Proxy is used
  - Align concepts with draft-amsuess-core-cachable-oscore

- Need for document reviews

Thank you!

Comments/questions?

https://gitlab.com/crimson84/draft-tiloca-core-observe-responses-multicast

Backup

Server side

1. Build a GET phantom request; Observe option set to 0

2. Choose a value T, from the Token space for messages …
   - … coming from the multicast IP address and addressed to target resource

3. Process the phantom request
   - As coming from the group and its IP multicast address
   - As addressed to the target resource

4. Hereafter, use T as token value for the group observation

5. Store the phantom request, with no reply right away

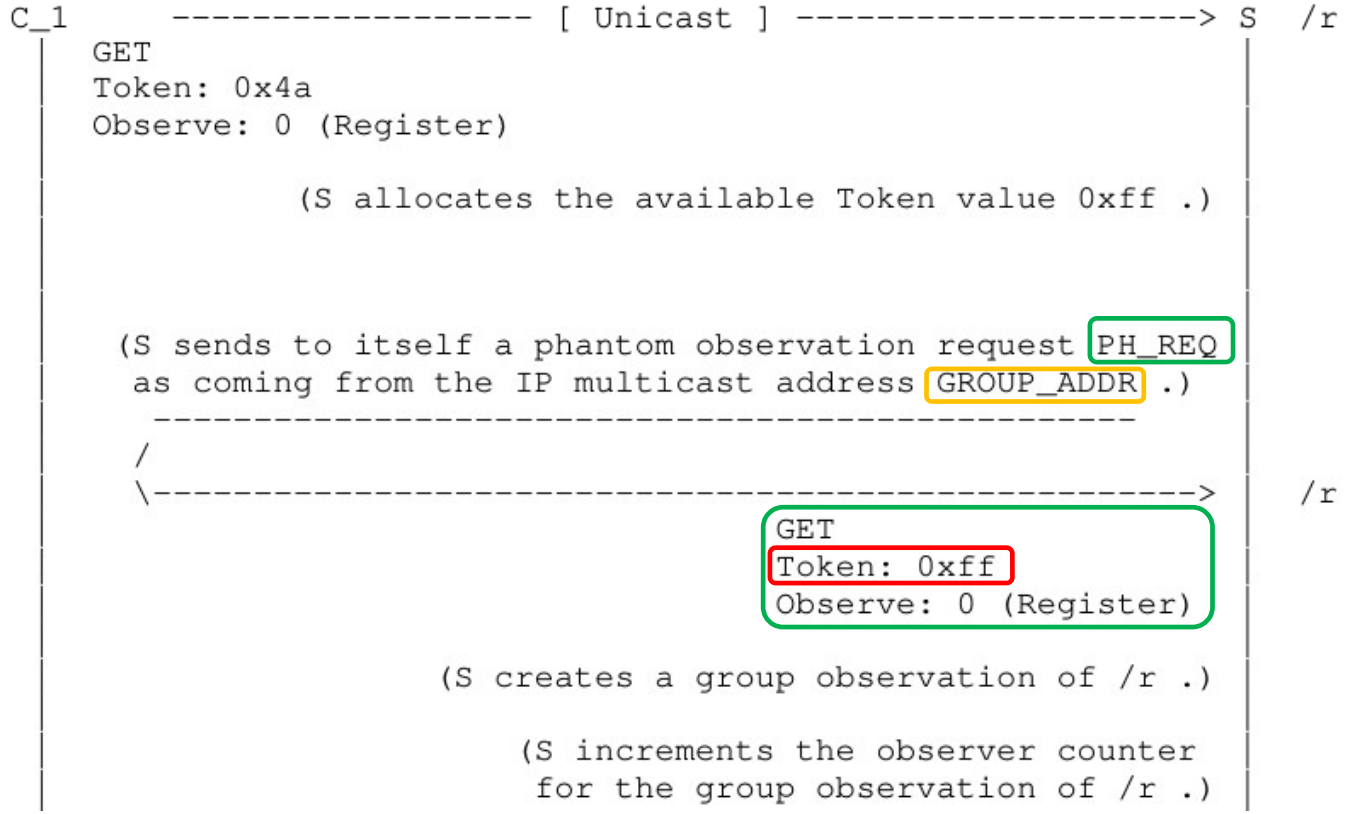
Interaction with clients

- The server sends to new/shifted clients an ***error response*** with
  - '*ph_req*': serialization of the phantom request
  - '*last_notif*': serialization of the latest sent notification for the target resource
  - '*cli_addr*' and '*cli_port*': source address/port of the phantom request
  - '*srv_addr*' and '*srv_port*': destination address/port of the phantom request

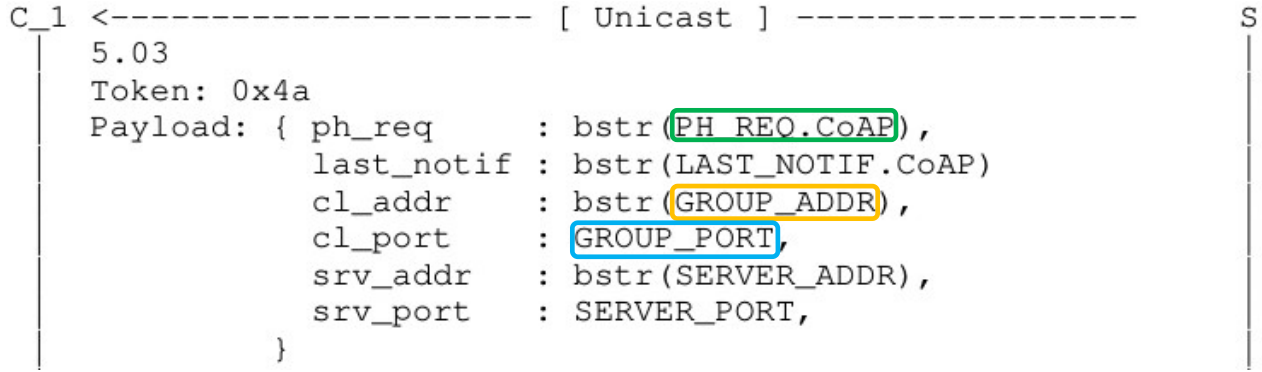
- When the value of the target resource changes:
  - The server sends an Observe notification to the IP multicast address '*cli_addr*'
  - The notification has the Token value T of the phantom request

- When getting the error response, a client:
  - Configures an observation for an endpoint associated to the multicast IP address
  - Accepts observe notifications with Token value T, sent to that multicast IP address

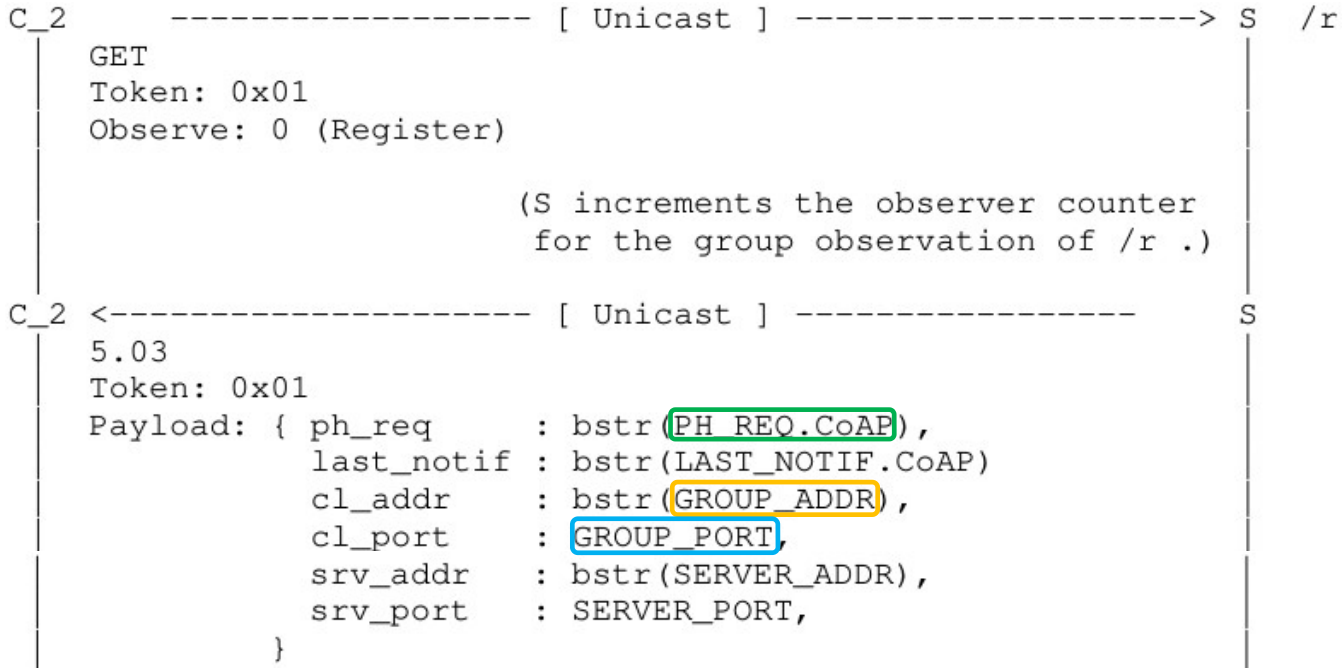
C1 registration



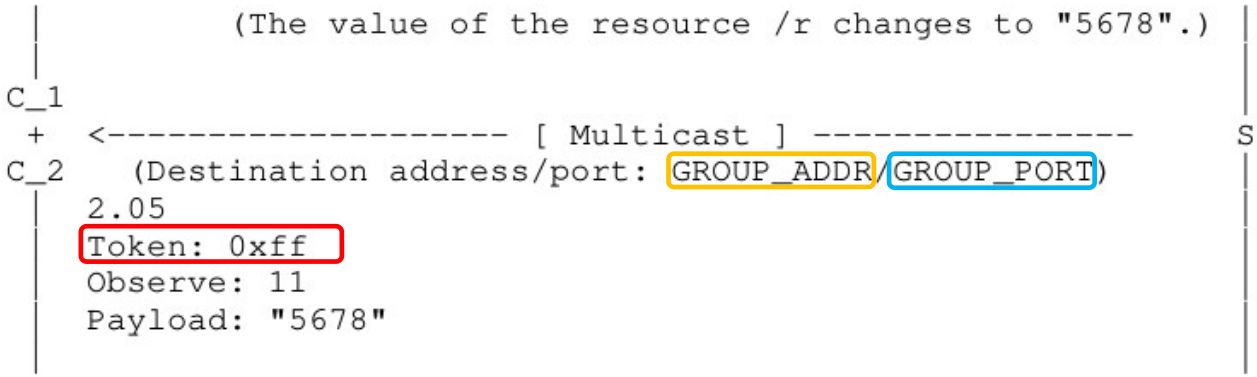
C1 registration



C2 registration



Multicast notification



- Same Token value of the Phantom Request
- Enforce binding between
  - Every multicast notification for the target resource
  - The (group) observation that each client takes part in

Security with Group OSCORE

- The phantom request is protected with Group OSCORE
  - ***x*** : the Sender ID ('kid') of the Server in the OSCORE group
  - ***y*** : the current SN value ('piv') used by the Server in the OSCORE group
  - Note: the Server consumes the value ***y*** and does not reuse it as SN in the group

- To secure/verify all multicast notifications, the OSCORE *external_aad* is built with:
  - 'req_kid' = ***x***
  - 'req_piv' = ***y***

- The phantom request is still included in the informative response
  - Each client retrieves ***x*** and ***y*** from the OSCORE option

Security with Group OSCORE

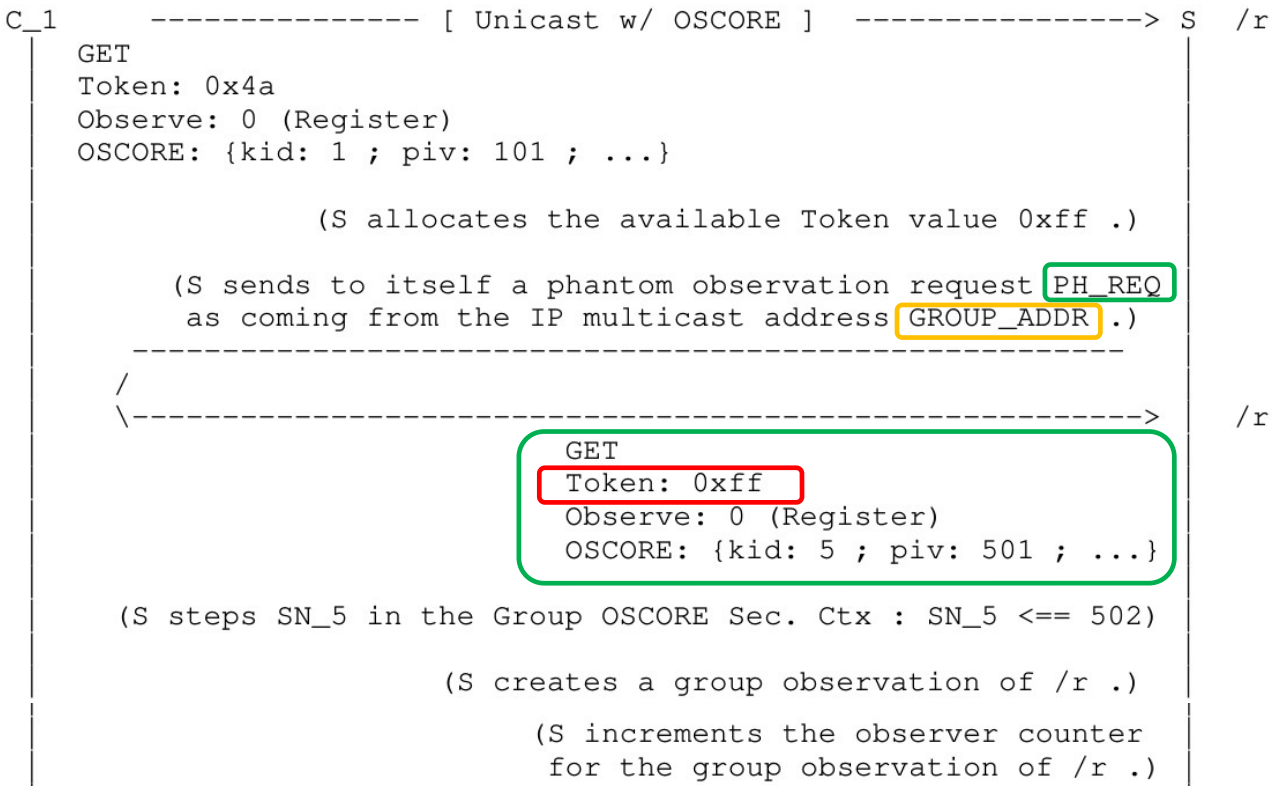
- In the error response, the server can ***optionally*** specify also:
  - '*join-uri*' : link to the Group Manager to join the OSCORE group
  - '*sec-gp*' : name of the OSCORE group
  - '*as-uri*' : link to the ACE Authorization Server associated to the Group Manager
  - '*cs-alg*' : countersignature algorithm
  - '*cs-alg-crv*' : countersignature curve of the algorithm
  - '*cs-key-kty*' : countersignature key type
  - '*cs-key-crv*' : countersignature curve of the key
  - '*cs-kenc*' : countersignature key encoding
  - '*alg*' : AEAD algorithm
  - '*hkdf*' : HKDF algorithm

MUST

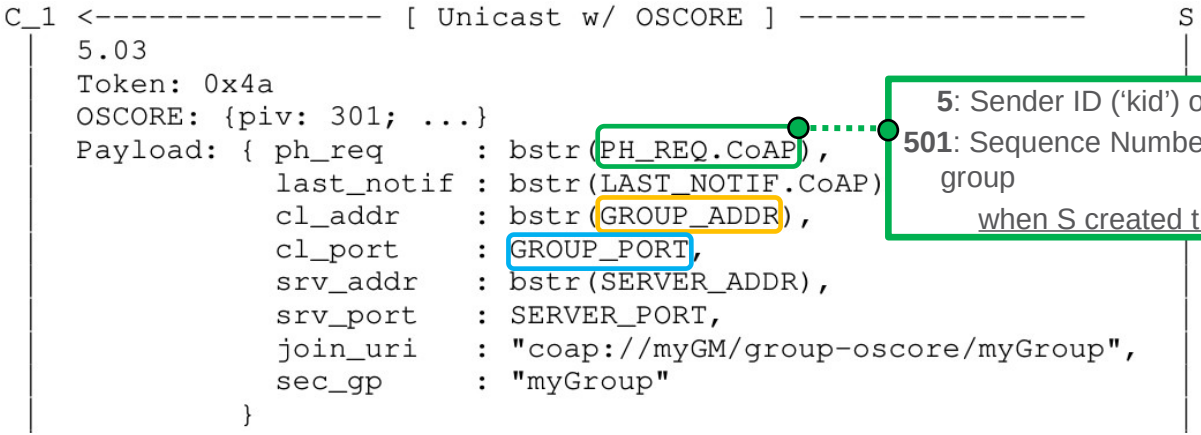
MAY

- Clients can still discover the OSCORE group through other means
  - E.g., using the CoRE Resource Directory, as in *draft-tiloca-core-oscore-discovery*

C1 registration w/ security



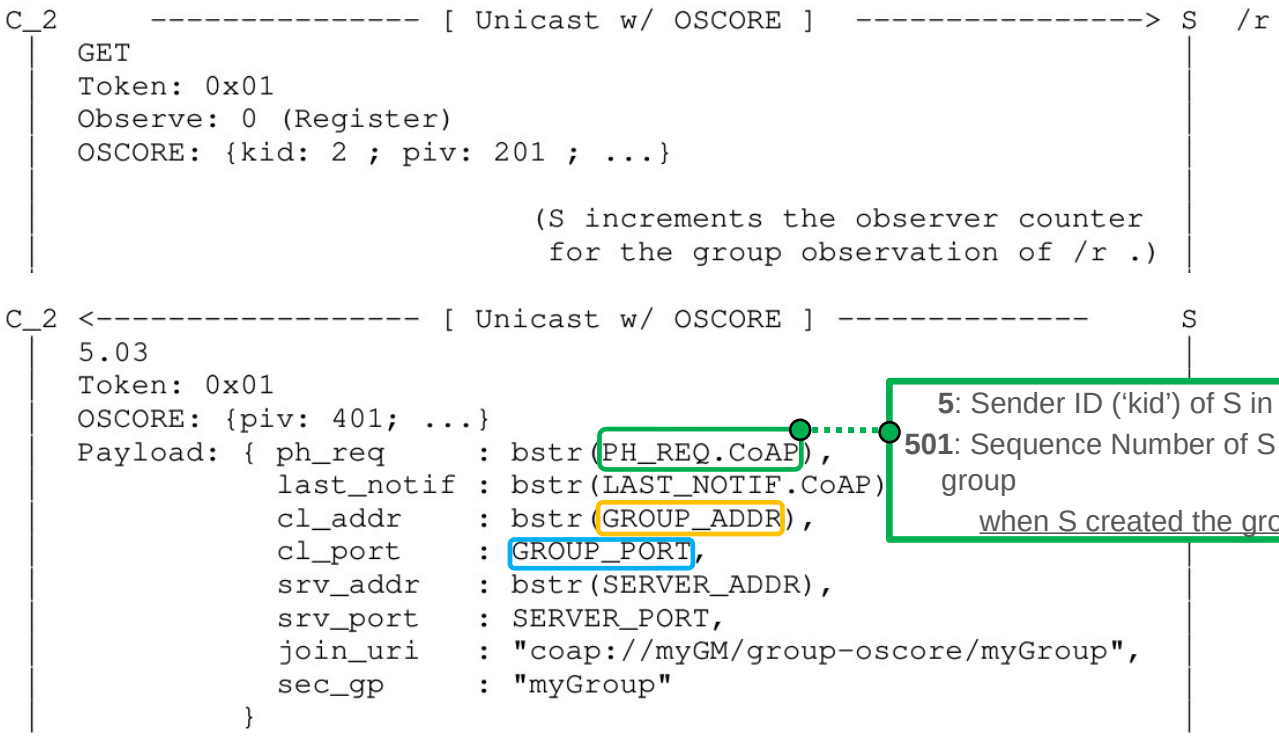
C1 registration w/ security



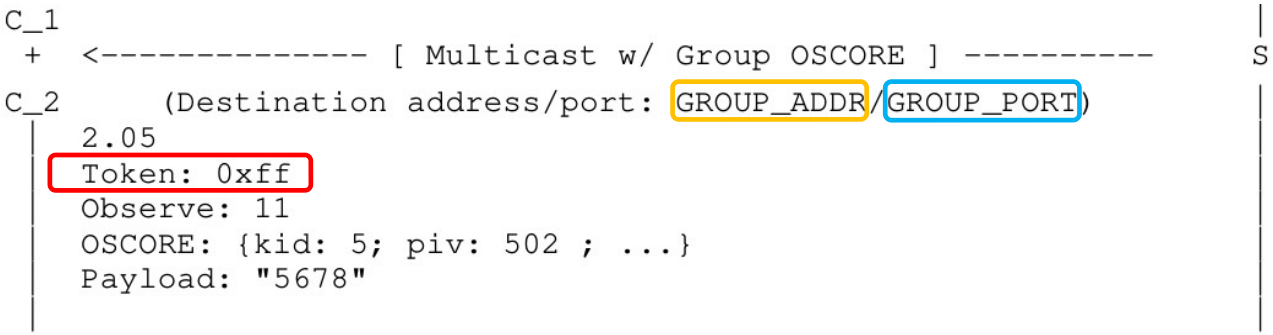
5: Sender ID ('kid') of S in the OSCORE group

501: Sequence Number of S in the OSCORE group when S created the group observation

C2 registration w/ security



Multicast notification w/ security



- When encrypting and signing the multicast notification:
  - The OSCORE *external_aad* has 'req_kid' = **5** and 'req_iv' = **501**
  - Same for all following notifications for the same resource
- Enforce secure binding between
  - Every multicast notification for the target resource
  - The (group) observation that each client takes part in