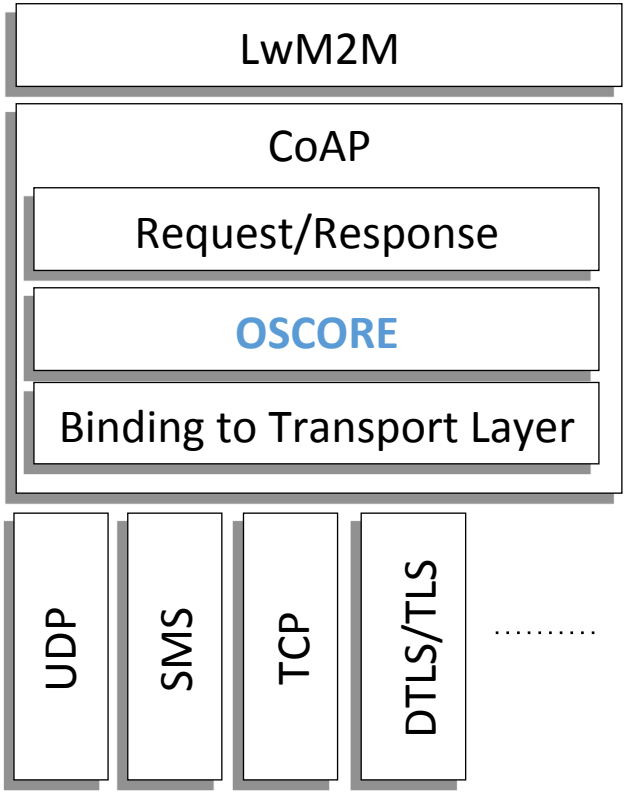
OSCORE for LwM2M (PR #105)

OMA DM interim 180207

Göran Selander, Ericsson

OSCORE overview

- Security protocol for CoAP defined in the IETF
- Designed for constrained networks
- Faithful to CoAP
  - Supports Observe (RFC 7641), Blockwise (RFC 7959), Patch & Fetch (RFC 8132), CoAP over TCP, …
- Independent of transport layer protocol
- Applicable to CoAP-mappable protocols
  - Protocol messages which can be mapped to a subset of to CoAP, e.g. HTTP, LwM2M
- Supports intermediate node operations
  - Proxy forwarding (RFC 7252)
  - Protocol translations, incl. HTTP-CoAP (RFC 8075)
  - Semi-trusted proxies (e.g. Read but not Write)

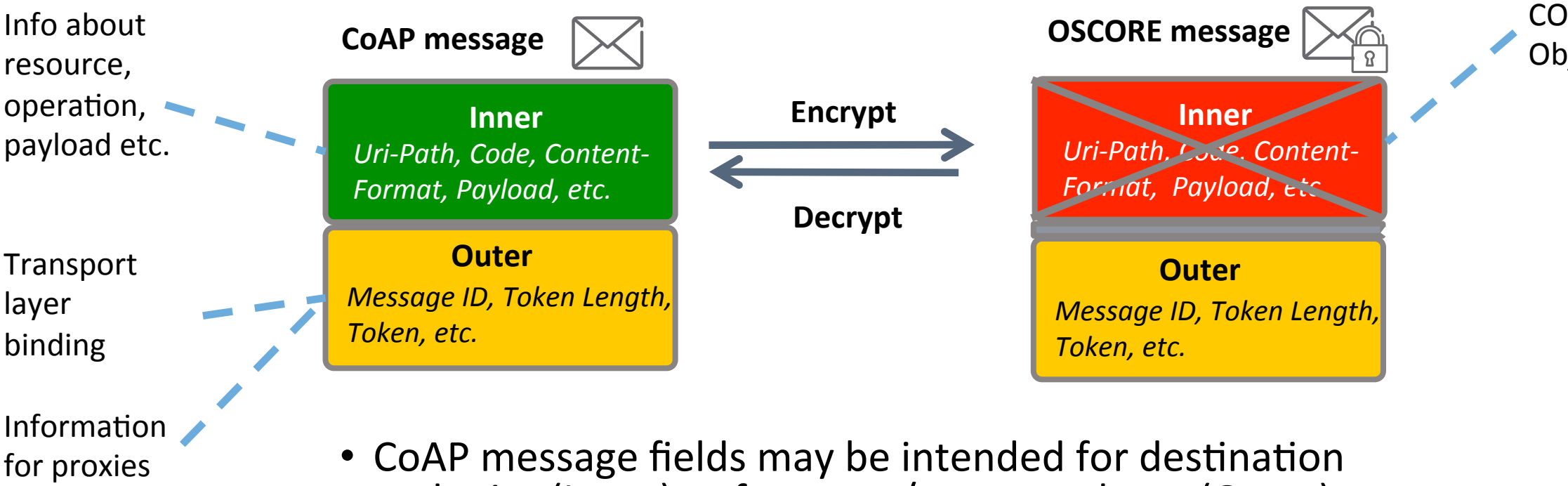


Basis for end-to-end secu

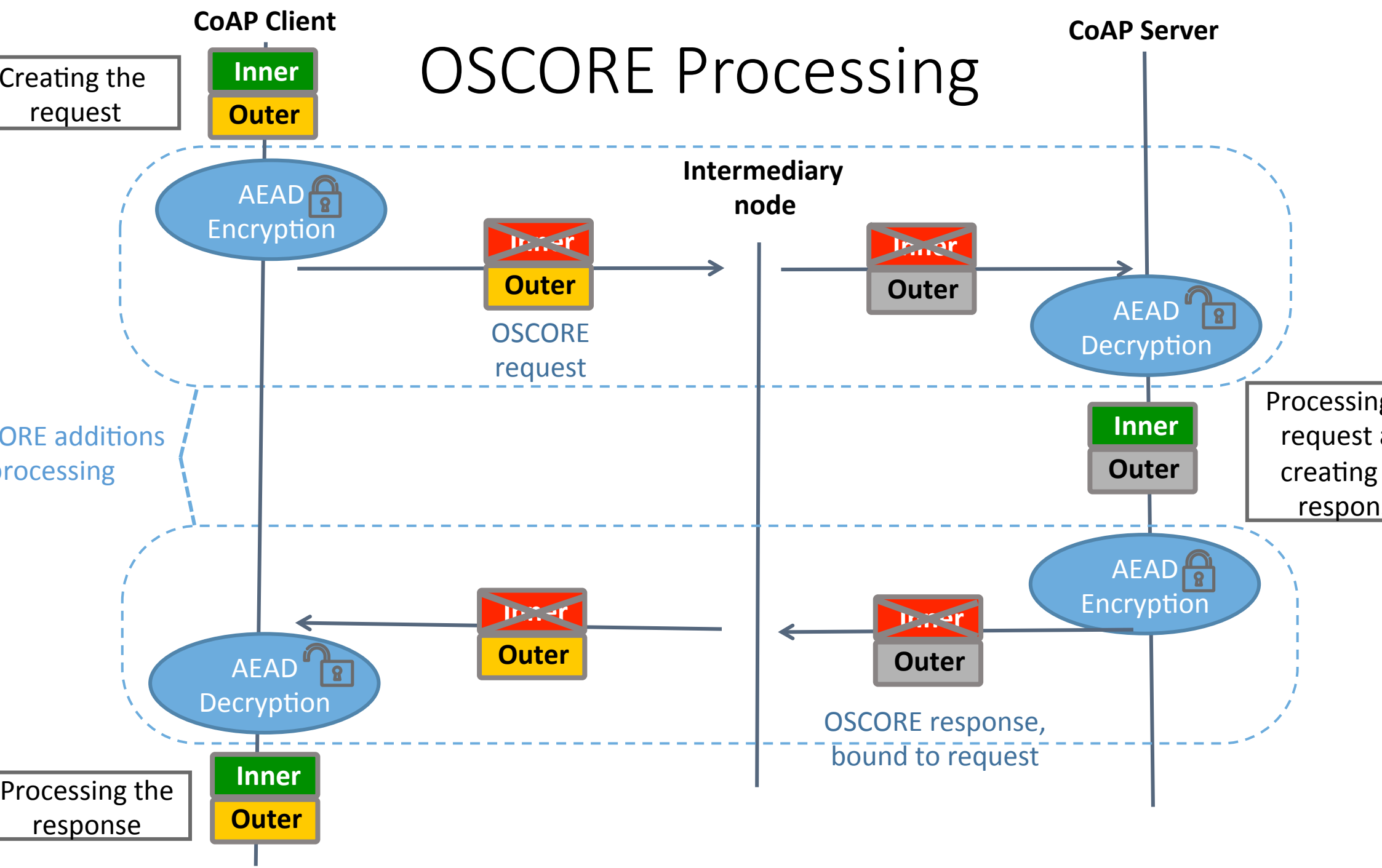
nd-to-End Security for Constrained Networks

- Security considerations
  - Intermediate nodes, proxy translations, mix of transports → untrusted proxies, dependence on transport
  - Unreliable or lossy networks → issues with replay, freshness and binding response to request
  - Security protocol message size, round trips → power consumption, latency
- Application layer protocol over transport layer security is not automatically secure
  - Initial CoAP suite (CoAP/Observe/Blockwise over DTLS) has vulnerabilities
  - CoAP security updates needed for LwM2M v1.1 [2]
  - If LwM2M is defined for new transports a careful security analysis is required
- OSCORE designed to provide end-to-end security for constrained settings
  - End-to-end security requirements for CoAP [3]
  - Complies with CoAP Security Update (PR #122), LwM2M v1.1 security requirements (PR #127)
  - Low message overhead, small footprint

OSCORE processing



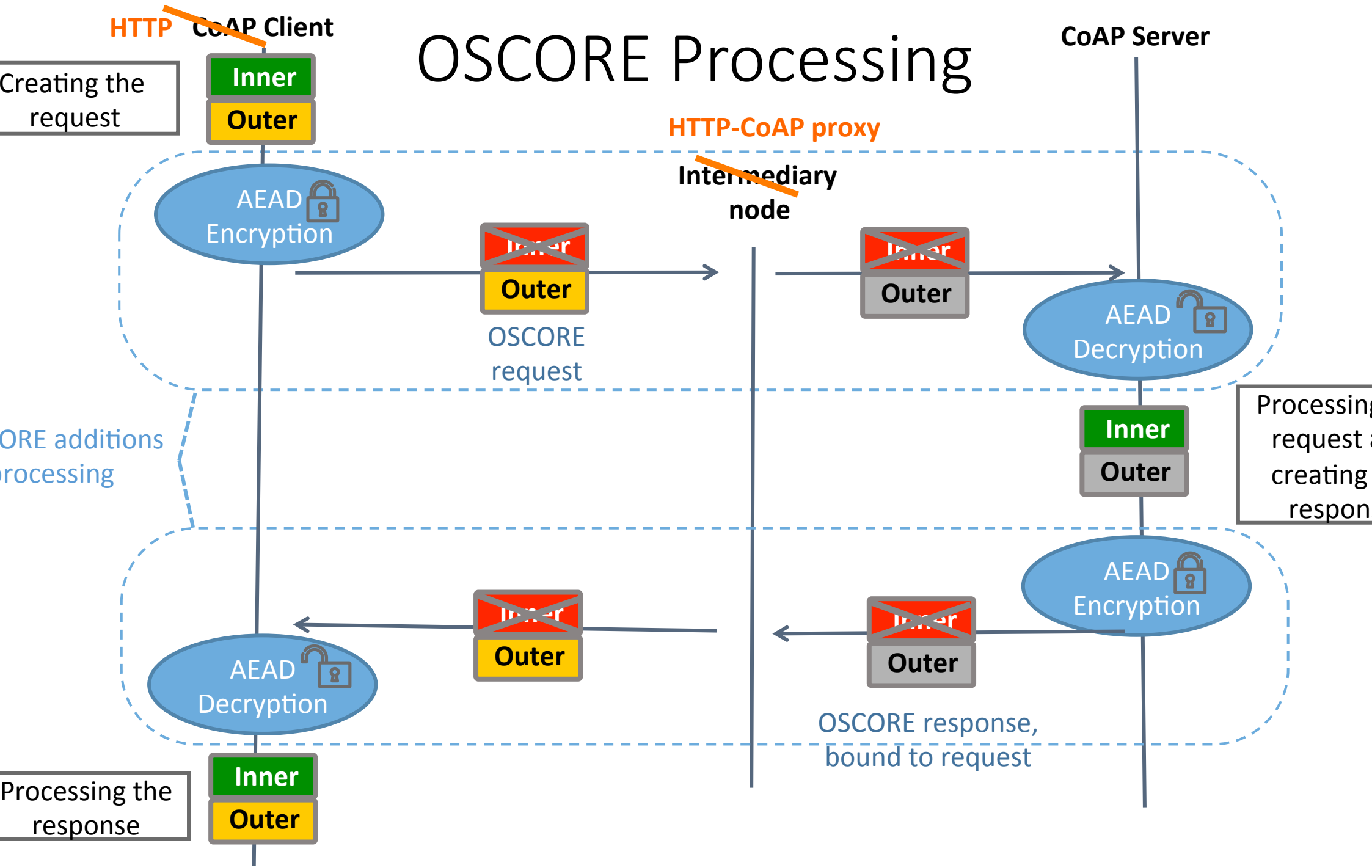
- CoAP message fields may be intended for destination endpoint (Inner) or for proxy/transport layer (Outer)
- Inner message fields are protected with COSE (RFC 8152)
- Uses Authenticated Encryption with Additional Data (AEAD)
  - AES-128-CCM-8 by default


OSCORE Processing
CoAP Client
CoAP Server
Intermediary node
Creating the request
Inner
Outer
AEAD Encryption
Outer
OSCORE request
Outer
AEAD Decryption
Inner
Outer
Processing request a creating respon
ORE additions processing
AEAD Encryption
Outer
Outer
AEAD Decryption
OSCORE response, bound to request
Processing the response
Inner
Outer

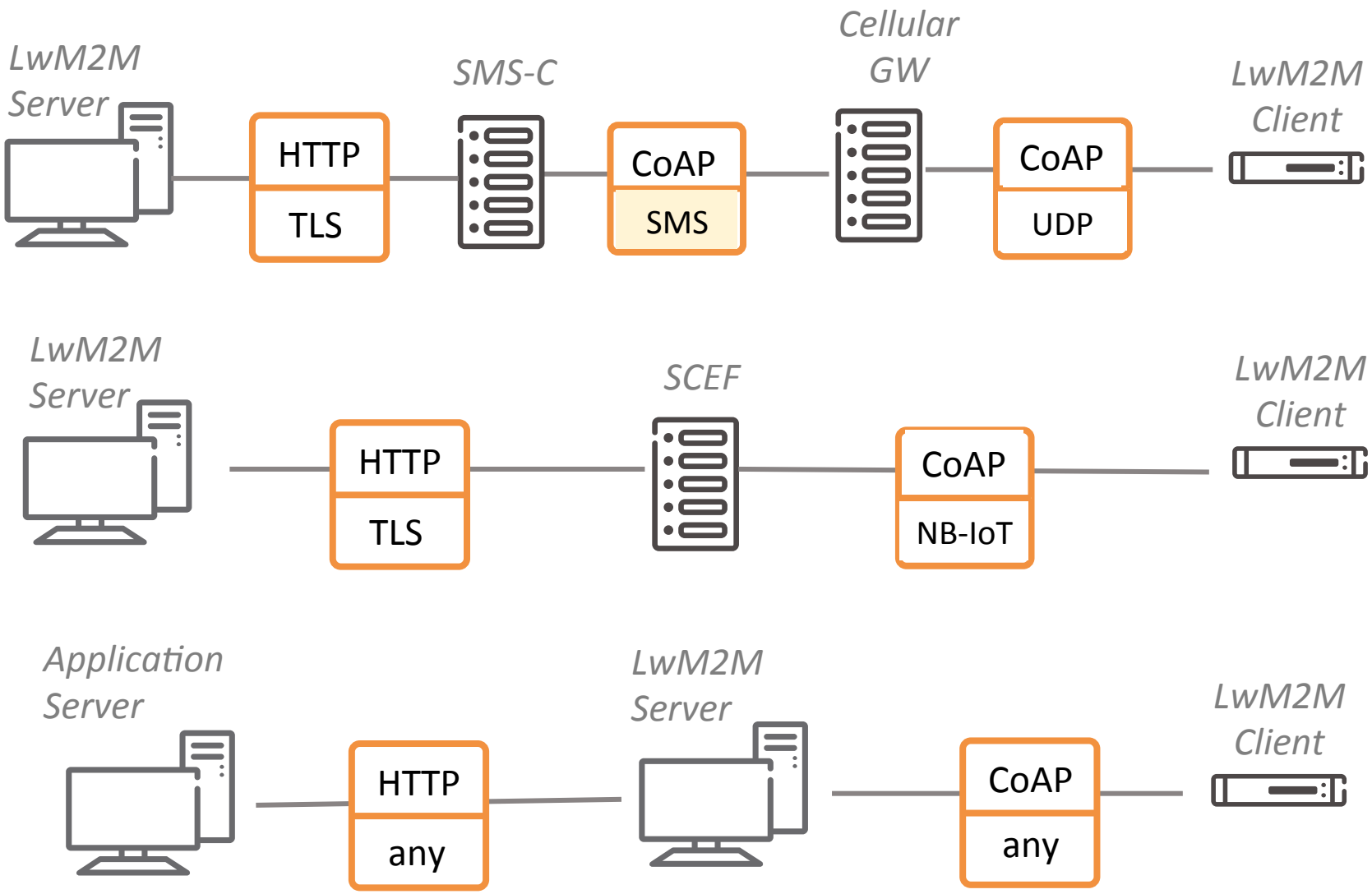
Transport in CoAP-mappable protocol messages

- The use of OSCORE is indicated with a new CoAP Option: Object-Security
- OSCORE messages can be natively "transported" in CoAP
  1. Add the CoAP Option Object-Security
  2. Put the COSE Object in the CoAP Payload
- The same procedure applies to other CoAP-mappable protocols
- OSCORE messages can be natively transported in HTTP
  1. Map HTTP message to CoAP
  2. Apply OSCORE to generate the COSE Object
  3. Add the new HTTP Header Field Object-Security
  4. Put the COSE Object in the HTTP Body

  Can be done in one step
- In the same way OSCORE messages can be natively transported in LwM2M
  - Not included in PR#105


OSCORE Processing
HTTP CoAP Client
CoAP Server
HTTP-CoAP proxy
Intermediary node
Creating the request
Inner
Outer
AEAD Encryption
Outer
OSCORE request
Outer
AEAD Decryption
Inner
Outer
Processing request creating respon
ORE additions processing
AEAD Encryption
Outer
Outer
AEAD Decryption
OSCORE response, bound to request
Processing the response
Inner
Outer

OSCORE End-to-End Security Examples



Can support differer
trust models, e.g.:

- Intermediary noc
  read but not wri
- Intermediary noc
  neither read nor

Implementation Support

- Several independent open source implementations available
  - Java (Californium): https://bitbucket.org/lseitz/oscoap_californium
  - C (Contiki, Erbium): https://github.com/Gunzter/contiki-oscoap
  - Python (aiocoap): https://github.com/chrysn/aiocoap
  - C# (CoAP-CSharp): https://github.com/Com-AugustCellars/CoAP-CSharp
  - Python (CoAP for openwsn): https://github.com/openwsn-berkeley/coap
  - C (openwsn-fw): https://github.com/openwsn-berkeley/openwsn-fw
- Interop tests at IETF Hackathon, ETSI F-Interop, Automated testing
  - https://ericssonresearch.github.io/OSCOAP/
  - https://github.com/malishav/TD-OSCORE
- Ongoing work
  - Integration in Californium 2.0.x (next step: Leshan)
  - Integration in Erbium (next step: Wakaama), Contiki NG
  - Wireshark dissector

  https://code.wireshark.org/review/#/q/topic:coap-oscore-preparation
  https://www.youtube.com/watch?v=05-MTLrFSrg
- Interest from Zephyr OS and RIOT OS

OSCORE in LwM2M

- Only minor change: Object-Security option + provisioning of OSCORE parameters
- OSCORE needs the following parameters:
  Master Secret, LwM2M Client ID, LwM2M Server ID (+ optional parameters)
- Since OSCORE is independent of DTLS/TLS, new Object Resources are needed
- Two alternatives:
  1. All OSCORE resources in the Security Object:

OSCORE Security Mode	Optional
OSCORE Master Secret	Optional
OSCORE LwM2M Client ID	Optional
OSCORE LwM2M ServerID	Optional
OSCORE ...	Optional, even if OSCORE is used

2. One OSCORE resource in the Security Object:

OSCORE Security Mode	Optional

+ a new "OSCORE Security Object" present only if OSCORE is used:

OSCORE Master Secret	Mandatory
OSCORE LwM2M Client ID	Mandatory
OSCORE LwM2M ServerID	Mandatory
OSCORE ...	Optional

The Resource "OSCORE Security Mode" decides if OSCORE is used or not

Summary

- OSCORE is a security protocol defined in the IETF
- Designed for end-to-end security in constrained settings
- Independent of transport layer protocol
- Applies to CoAP-mappable protocol messages (used by OCF)
- OSCORE also enables multicast security (supported by Fairhair alliance)
- Supports proxy operations such as stateless forwarding (used by 6tisch/ IEEE 802.15.4e)
- Good implementation support: Several independent open source implementations, Wireshark support, integration in Californium and Erbium in progress

References

[1] https://tools.ietf.org/html/draft-ietf-core-object-security

[2] https://tools.ietf.org/html/draft-ietf-core-echo-request-tag

[3] https://tools.ietf.org/html/draft-hartke-core-e2e-security-reqs

[4] https://tools.ietf.org/html/draft-mattsson-core-coap-actuators