

# Distribution of Software Updates with End-to-End Secure Group Communication and Block-Wise Transfer for CoAP

*draft-tiloca-t2trg-sw-update-groupcomm-00*

**Marco Tiloca, RISE**

IETF 123 Meeting – Madrid – July 21<sup>st</sup>, 2025

# Motivation

- › **During their lifetime, IoT devices have to receive and install SW updates**

- Add and extend features; address security vulnerabilities; ...
- The process has to be secure, but also efficient and scalable



- › **Lot of work on SW/FW update, especially in the IETF SUIT WG**

- Architecture (RFC 9019) and Manifest information model (RFC 9124)
- Manifest serialization in CBOR (*draft-ietf-suit-manifest*, in the RFC Editor Queue)

- › **Lot of work on secure (group) communication in the IETF CoRE WG**

- › **Can we use building blocks from CoRE to make SW update more efficient?**

- Compatible with (but not dependent on) the SUIT architecture and Manifest
- Main focus on protection and actual distribution of the SW update

# Contribution

- › **Method for efficiently distributing a software update to multiple target devices**

- Architecture
- Advertisement of available SW update
- One-to-many distribution of SW update

- › **Main building blocks**

- CoAP
- Block-wise transfers for CoAP
- End-to-security security with Group OSCORE

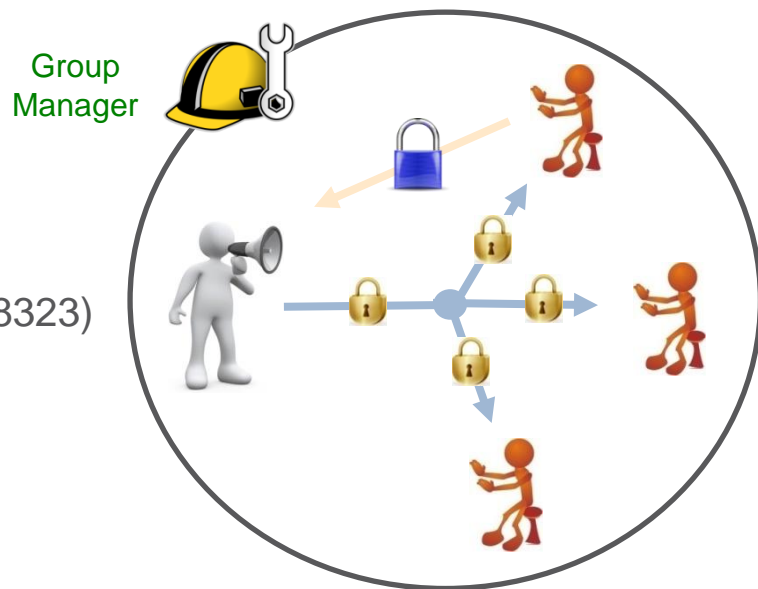
- › **Early idea presented at the T2TRG interim meeting in May 2024**

<https://datatracker.ietf.org/meeting/interim-2024-t2trg-01/materials/slides-interim-2024-t2trg-01-sessa-distribution-of-software-updates-with-end-to-end-secure-group-communication-for-coap-00>

# Building blocks from the CoRE WG

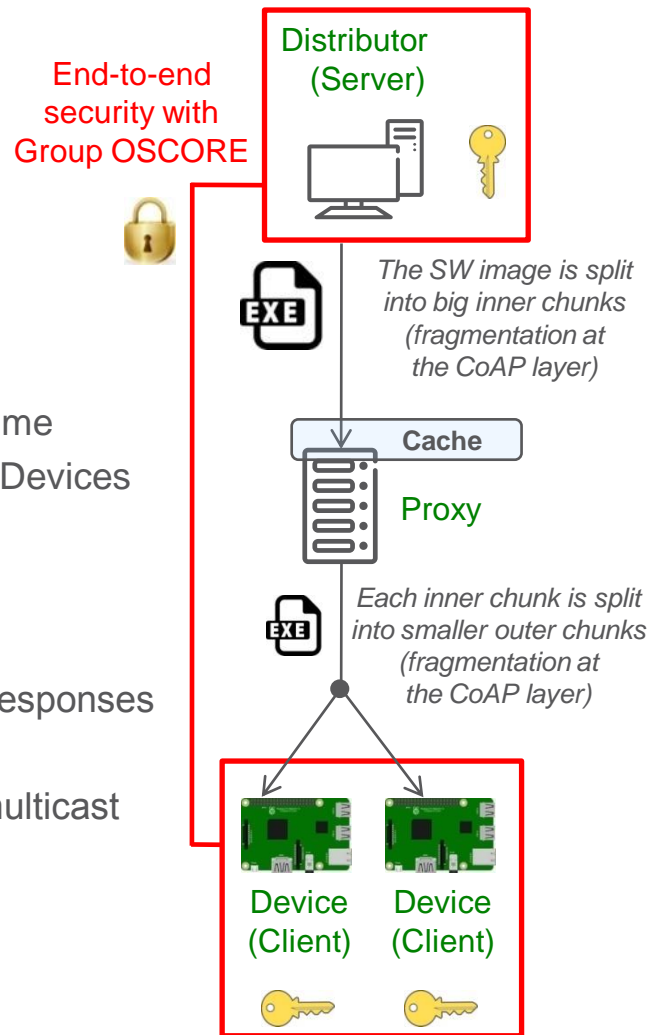
## Protocols and concepts on lightweight secure (group) communication

- › **Web-transfer protocol CoAP (RFC 7252)**
- › **Notable extensions**
  - Observe (RFC 7641)
  - Block-wise (RFC 7959)
  - CoAP and Block-wise over reliable transports (RFC 8323)
- › **End-to-end security with OSCORE (RFC 8613)**
- › **CoAP group communication**
  - Group OSCORE for end-to-end security in groups
  - Cacheability of end-to-end protected responses
  - One-to-many responses as Observe multicast notifications



# Approach overview

- › All the Devices want the same thing!
- › One-to-many distribution, with CoAP responses
  - Sent over multicast to the Devices, from an advertised point in time
  - Protected end-to-end, between the Distributor and the recipient Devices
- › Delivered by an intermediary CoAP proxy that:
  - Fetches large “inner” chunks from the Distributor, as protected responses
  - Caches those protected responses generated by the Distributor
  - Sends small “outer” chunks to the Devices as responses over multicast



# Design goals

## 1. End-to-end security between Devices and Distributor

- Use of Group OSCORE, with all the Devices and the Distributor in the same OSCORE group
  - › Seamless distribution of keying material through the OSCORE Group Manager
  - › If the OSCORE group is for this single application, then group membership implies authorization

## 2. Limit interactions and exchanges with the Distributor

- Proxy deployed between Devices and Distributor
- Large-size inner chunks from the Distributor to the Proxy, through Block-wise over reliable transport (BERT)
- End-to-end protected responses generated by the Distributor are cached at the Proxy

## 3. Limit interactions and exchanges with the Devices

## 4. Avoid lockstep Block-wise transfers to the Devices; allow recovery of lost blocks

- End-to-end protected responses generated by the Distributor are cached at the Proxy
- Block-wise multicast responses to the Devices, reusing concepts defined for Observe multicast notifications

## 5. Accommodate constrained links used by the Devices

- Use of small-size outer chunks between Proxy and Devices

# Architecture

## > Author

- Creates the SW image and the manifest with metadata
- Uploads the image and manifest to the Distributor

## > Distributor

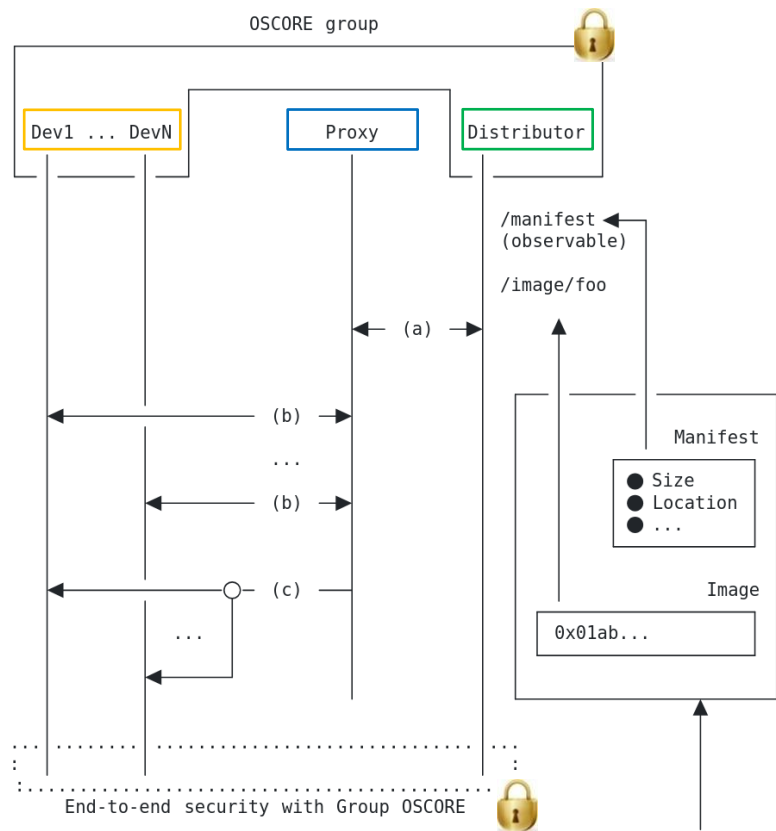
- Hosts the image and manifest in appropriate resources
- Distributes those to the Devices through the Proxy

## > Proxy

- Caches image and manifest obtained from the Distributor
- Obtains large inner chunks of the image from the Distributor
- Provides small outer chunks of the image to the Devices

## > Devices

- Obtain the latest manifest through the Proxy (notifications)
- Enroll in the group distribution of the latest image



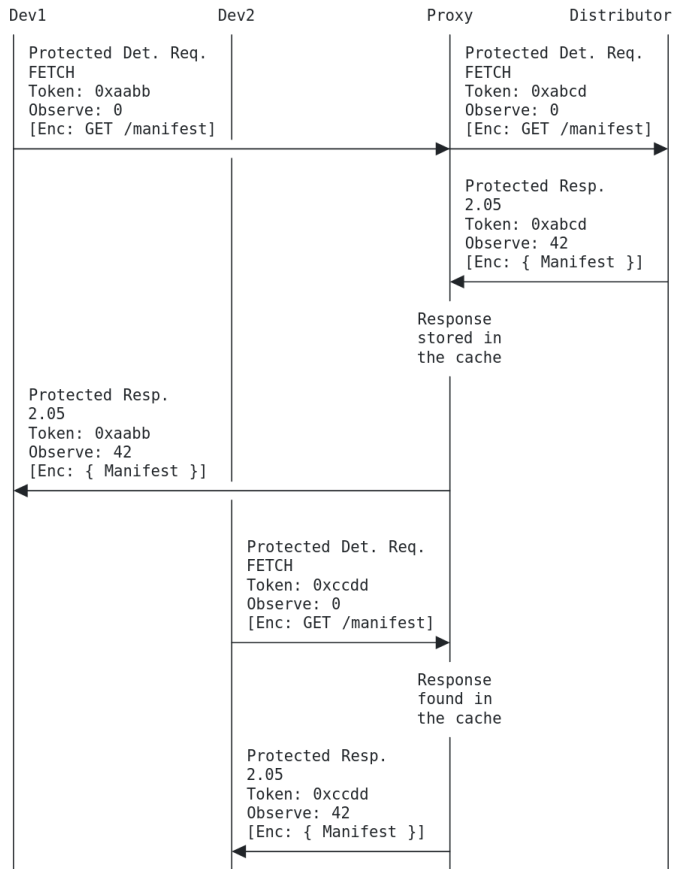
(a) CoAP over TCP, using BERT to transfer large inner chunks of the image

(b) CoAP over UDP

(c) CoAP over UDP and IP multicast, using Block-wise to transfer small outer chunks of the image

# Notification of new SW update

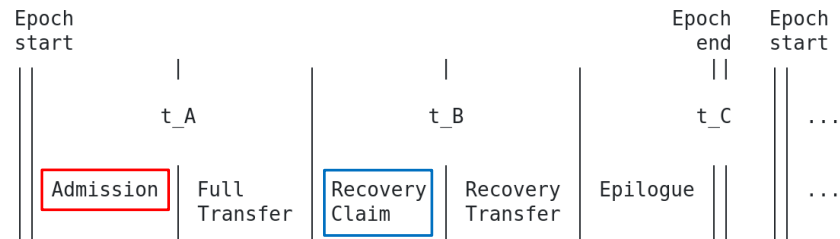
- › **The Distributor hosts the manifest**
  - At an observable resource for the SW component in question
- › **The Proxy caches responses from the Distributor**
  - All the Devices in the OSCORE group produce the same protected Deterministic Request, which results in a cache hit at the Proxy (*draft-amsuess-core-cachable-oscore*)
  - The Proxy has limited interactions with the Distributor (e.g., first request ever for the manifest of that SW component)
  - The later image distribution exploits the same principle ...
- › **Possible optimization**
  - The Distributor can use Observe multicast notifications
  - Tell multiple Proxies (thus their Devices) about the new manifest
  - *draft-ietf-core-observe-multicast-notifications*



# Distribution of SW update (1/3)

## › Organized into epochs by the Proxy

- One epoch is used to distribute one large inner chunk of the image, with index INNER\_INDEX
- Circular: once transferred the last inner chunk, the next epoch is for transferring the first inner chunk



## › A Device can start acquiring the image at any epoch

- It takes as many epochs as needed for obtaining the whole image
- It can actually “enroll” in an epoch only during the “Admission” phase
- It can ask for missed outer chunks during the “Recovery claim” phase

## › Through error responses, the Proxy gives guidance about:

- How to receive the image, as delivered in small outer chunks
- The overall progress of the image transfer (i.e., the current INNER\_INDEX)
- Notable checkpoints and ending of the current epoch (determined dynamically)



# Distribution of SW update (2/3)

## › Kick-off for the proxy

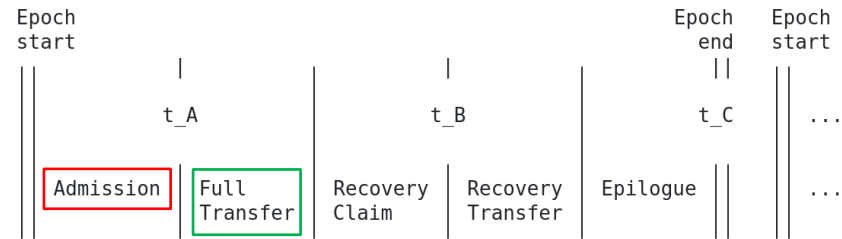
- Received a Deterministic Request and no cache hit
- Retrieve the first inner chunk from the Distributor
- Start the epoch for INNER\_INDEX = 0 and set  $t_A$

## › “Admission” phase

- A Device can enroll to receive the inner chunk of the current epoch
- The Device sends a Block-wise request, with Block2: NUM=0, M=0, SZX=2
- The Proxy replies with a 5.03 error response including:
  - › “tp\_info”: transport-specific information, with details for listening to multicast responses
  - › “next\_not\_before”: time left until  $t_A$  when the “Full transfer” phase starts
  - › “progress\_indicator”: value of INNER\_INDEX for the current epoch

## › “Full transfer” phase

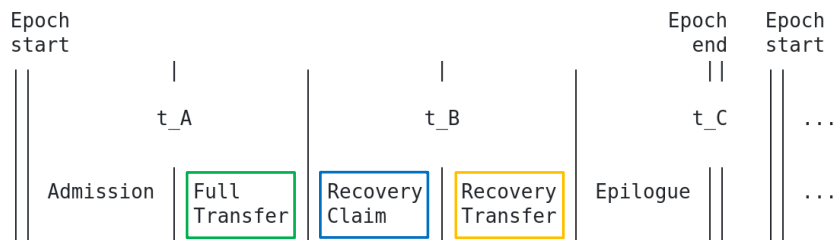
- The Proxy determines the time  $t_C$  when the current epoch will end
- The Proxy sends a sequence of Block-wise responses to the Devices, over IP multicast
- Each response is a small outer chunk of the large inner chunk for the current epoch
- Reject requests from Devices; the error response includes the CoAP Max-Age option, with value the time left until  $t_C$



# Distribution of SW update (3/3)

## › “Recovery claim” phase

- The Proxy determines  $t_B$  when this phase ends
- A Device can ask for outer chunks that it missed during the “Full transfer” phase of the current epoch
- A Device sends one request per missed outer chunk, indicated by NUM in the CoAP Block2 option
- The Proxy replies with an error response including:
  - › “tp\_info”: transport-specific information, without details for listening to multicast responses
  - › “next\_not\_before”: time left until  $t_B$  when the “Recovery transfer” phase starts
  - › “progress\_indicator”: value of INNER\_INDEX for the current epoch
  - › CoAP Max-Age option: time left until  $t_C$



## › “Recovery transfer” phase – Only meant for Devices that took the previous “Full transfer” phase

- The Proxy sends a sequence of Block-wise responses to the Devices, over IP multicast
- Each response is a small outer chunk requested by at least one Device in the previous “Recovery claim” phase
- Reject requests from Devices; the error response includes the CoAP Max-Age option, with value the time left until  $t_C$

## › Epilogue: the Proxy updates INNER\_INDEX (incremented, or set to 0 if the first inner chunk is next)

# Outer chunk checksum (1/2)

## › When a Device has received the whole SW image ...

- The Device can verify its integrity/authenticity, as generated by the intended Author
- E.g., based on a signature on the (hash of the) image included in the early manifest

## › When a Device has received an inner chunk **INNER\_X** ...

- The Device can verify its integrity/authenticity, as generated by the intended Distributor
- Based on the signature included when using Group OSCORE

## › When a Device has received an outer chunk **OUTER\_X\_Y** ... ?

- If no check is possible, it is very easy to perform an attack against availability
- An on-path adversary can inject a single corrupted outer chunk
- The Devices notice only when failing to verify the Group OSCORE signature on **INNER\_X**
- The adversary takes advantage of multicast to make the attack more efficient!

# Outer chunk checksum (2/2)

## › Mitigation and deterrent

- The Proxy computes a checksum `CHK_X_Y` on each outer chunk `OUTER_X_Y`
- The Proxy adds `CHK_X_Y` in the multicast response conveying `OUTER_X_Y`
  - › As value of a new CoAP “Checksum” option
- Devices can verify the integrity of `OUTER_X_Y` and treat the chunk as lost if that fails

## › The checksum is a 2-byte truncated MAC

- MAC input: the multicast response conveying `OUTER_X_Y`
- MAC key: derived using the Group OSCORE Security Context between Distributor and Devices
  - › From the same Master Secret and Master Salt, with the usual HKDF
  - › The “info” structure used for key derivation includes the OSCORE Partial IV of the multicast response
  - › The Partial IV is guaranteed to be always present in that response
- The same MAC key is used for all the `OUTER_X_Y` of the same `INNER_X`

## › Devices can always derive the MAC key; what about the Proxy?

# Providing the MAC Key to the Proxy

- › **The Distributor provides the Proxy with the MAC key**
  - In the response conveying `INNER_X`, as prepended to the protected CoAP payload
- › **Defined new CoAP “Pre-OSCORE-Data” option**
  - Value: uint identifier of the semantics of a CBOR data item prepended to the CoAP payload
  - Odd value X: the item is a CBOR byte string and has semantics X
  - Even value X+1: the item is a COSE object (RFC 9052) and has semantics X
- › **For the method defined in this document**
  - Option value: 1 → The CBOR byte string has as value the MAC key
    - › Preferable if the Distributor and the Proxy share a secure communication channel
  - Option value: 2 → The COSE Object conveys the MAC key using HPKE (RFC 9180) in COSE
    - › Required if the Distributor and the Proxy do not share a secure communication channel
    - › The Distributor needs to know the static public key of the Proxy
- › **Maybe the “Pre-OSCORE-Data” option and its general use can be defined in the CoRE WG?**

# Next steps

- › **Examples of message exchange**
- › **Approaches for the Distributor to learn and select the public key of the Proxy**
- › **Security considerations, beyond the inherited ones**
- › **As to recovering missed blocks (outer chunks), functional comparison with:**
  - Flute – <https://datatracker.ietf.org/doc/rfc3926/>
  - Norm – <https://datatracker.ietf.org/doc/rfc5740/>
- › **Comments are welcome!**

# Thank you!

<https://gitlab.com/crimson84/draft-tiloca-t2trg-sw-update-groupcomm>

# References

- › <https://www.rfc-editor.org/rfc/rfc7252.html>
- › <https://www.rfc-editor.org/rfc/rfc7641.html>
- › <https://www.rfc-editor.org/rfc/rfc7959.html>
- › <https://www.rfc-editor.org/rfc/rfc8323.html>
- › <https://www.rfc-editor.org/rfc/rfc8613.html>
- › <https://datatracker.ietf.org/doc/draft-ietf-core-groupcomm-bis/>
- › <https://datatracker.ietf.org/doc/draft-ietf-core-oscore-groupcomm/>
- › <https://datatracker.ietf.org/doc/draft-amsuess-core-cachable-oscore/>
- › <https://datatracker.ietf.org/doc/draft-ietf-core-observe-multicast-notifications/>
  
- › <https://www.rfc-editor.org/rfc/rfc9200.html>
- › <https://datatracker.ietf.org/doc/draft-ietf-ace-key-groupcomm-oscore/>
  
- › <https://www.rfc-editor.org/rfc/rfc9180.html>
- › <https://datatracker.ietf.org/doc/draft-ietf-cose-hpke/>
  
- › <https://www.rfc-editor.org/rfc/rfc9019.html>
- › <https://www.rfc-editor.org/rfc/rfc9124.html>
- › <https://datatracker.ietf.org/doc/draft-ietf-suit-manifest/>
- › <https://datatracker.ietf.org/doc/draft-ietf-suit-firmware-encryption/>