Object Security for the IoT

with focus on COSE over CoAP

Francesca Palombini, Göran Selander, Ericsson
RIOT Summit, July 16, 2016

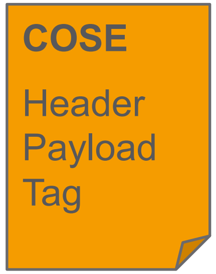
Outline

- Background (Göran)
  - COSE
  - COSE over CoAP
    - OSCOAP
    - EDHOC
    - ACE
- OSCOAP implementation (Francesca)

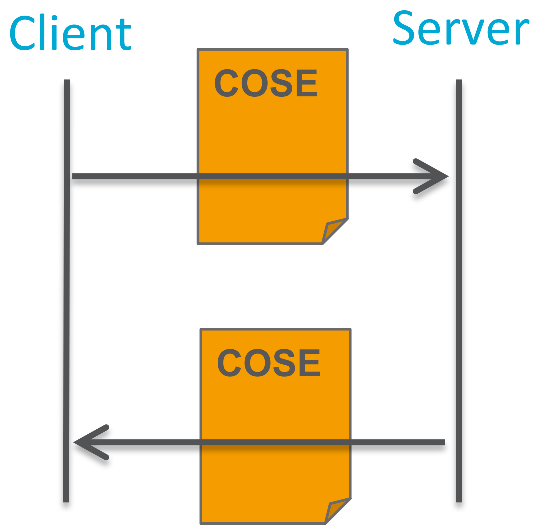
Background

COSE

- COSE = CBOR (RFC 7□49) Object Signing and Encryption
- Generic compact security wrapper
- COSE structure (simplified)
  - Header (protected, unprotected)
  - Payload
  - Tag (Signature/Message Authentication Code)

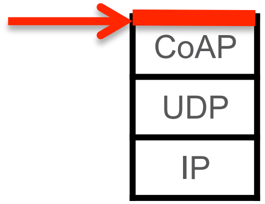


- Cryptographic protection of payload and protected header
- Internet draft, draft-ietf-cose-msg, passed WG last call
- Implementations in C, C#, Java
- This presentation is about security protocols based on COSE objects in CoAP messages
- COSE Payload examples:
  - Application data
  - CoAP message (CoAP payload, options, header fields)
  - Authorization information (“Access tokens”)
  - . . .

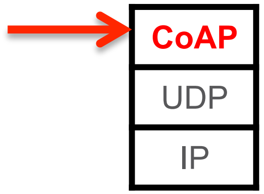


Security on different layers

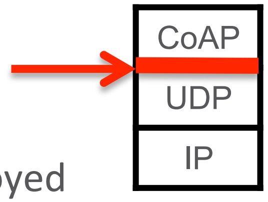
- Protection of application data
  - applies to different transfer protocol (CoAP/HTTP/ …)
  - end-to-end security across proxies

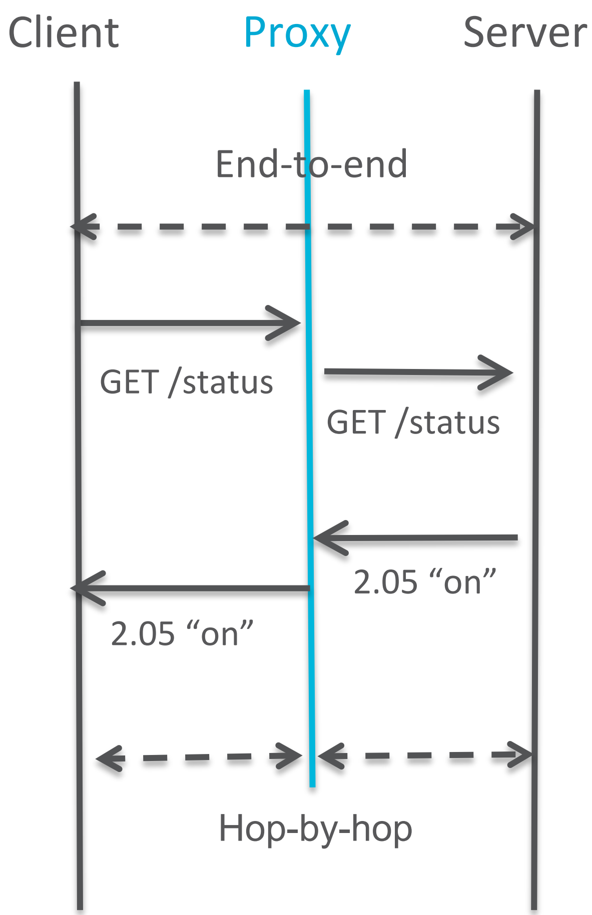


- Protection of CoAP message
  - applies to different underlying layers (TCP/UDP/IP/ …)
  - protects CoAP message semantics
  - end-to-end security aligned with proxy functionality



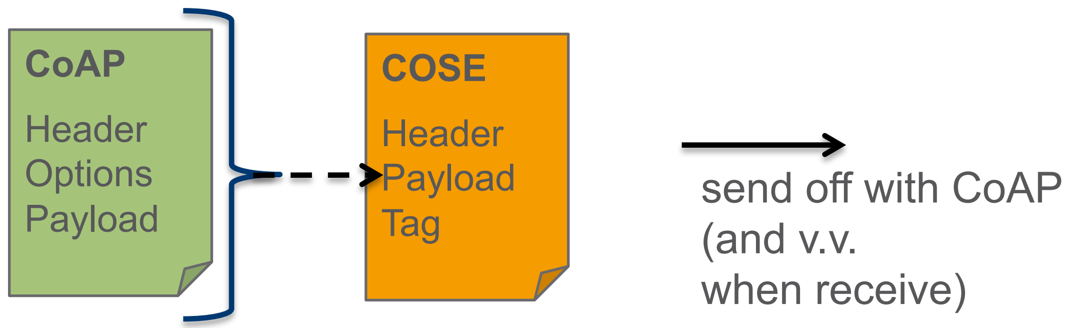
- Protection of transport layer
  - TLS/DTLS well known and widely deployed
  - hop-by-hop security, message unprotected in proxy





- End-to-end security requirements: [draft-hartke-core-e2e-security-reqs]

OSCOAP – protecting CoAP message



- OSCOAP protects as much as possible of CoAP messages
- Encryption, integrity and replay protection of CoAP payload, almost all options, and selected header fields
  - Not Uri-Host, Uri-Port, Proxy-Uri,Proxy-Scheme
  - Supports Block options
- Provides CoAP in-layer security
  - Independent of how CoAP is transported (UDP, TCP, 802.15.4 IE, Bluetooth, foo…)
- Supports proxy forwarding
- Does not support proxy caching
- draft-selander-ace-object-security
- Current version -05

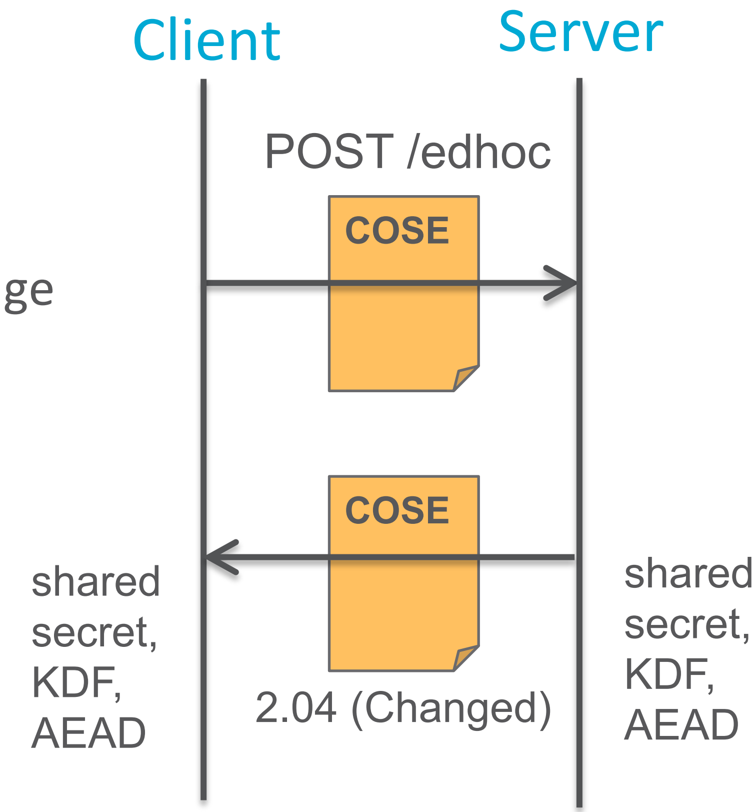
Example

```
Client                                                 Server
   |  request:                                            |
   |    GET example.com                                   |
   |    [Header, Token, Options:{...,                     |
   |     Object-Security:COSE object}]                    |
   +------------------------------------------------->|
   |  response:                                           |
   |    2.05 (Content)                                    |
   |    [Header, Token, Options:{...,                     |
   |     Object-Security:-}, Payload:COSE object]         |
   |<-------------------------------------------------+
   |                                                      |
```

Figure 1: Sketch of OSCOAP

EDHOC – establishing keys

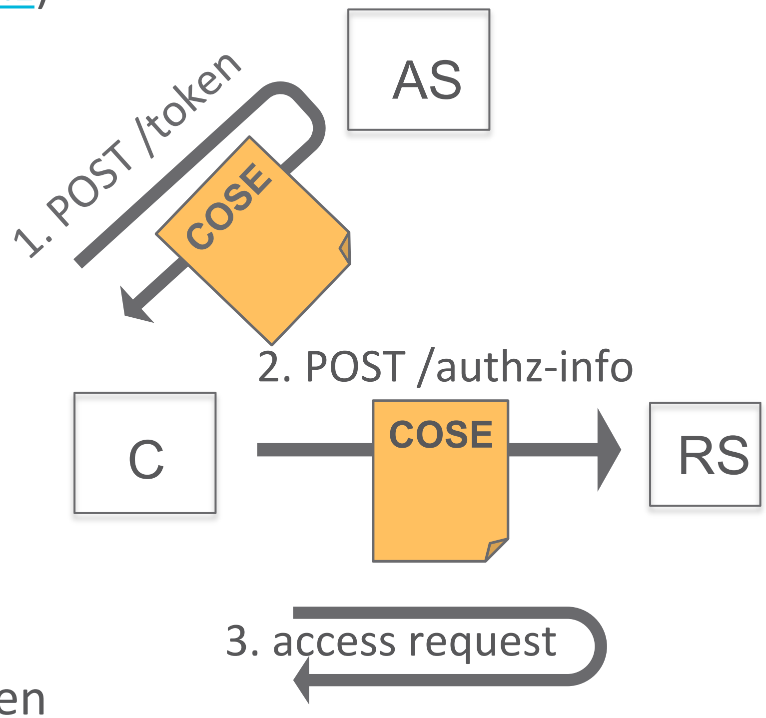
- Key establishment over COSE
- Ephemeral Diffie-Hellman over COSE (EDHOC)
- Mutual authentication based on pre-established credentials
  - Pre-shared keys
  - Raw public keys
  - X.509 certificates
- Forward secrecy: Diffie-Hellman key exchange
- Crypto agility: Algorithm negotiation
- Example of message sizes
  with PSK: 70-80 bytes, with RPK: 130-140 bytes
- With COSE in place, EDHOC comes at very little footprint
- May be implemented as CoAP POST
- [draft-selander-ace-cose-ecdhe]



ACE – authorization of access

- Authorization for access to IoT resources
- ACE Framework ([draft-ietf-ace-oauth-authz]())
- Based on OAuth 2.0

1. Client request access token from Authorization Server
2. Client forward access token to Resource Server
3. Client request access to Resource Server



- Keys can also be provisioned with the token
- Different ACE profiles to support different C-to-RS communication and security settings
- OSCOAP profile: [draft-seitz-ace-oscoap-profile]()

OSCOAP implementation

Work in progress; based on version -04

Java implementation

- Californium: a CoAP Java implementation*
- OSCOAP: patch for Californium, easy to maintain
- Dependencies: COSE Java implementation (that uses CBOR and tinyDTLS)

* http://www.eclipse.org/californium/

C implementation

- Erbium CoAP: a CoAP library in Contiki OS*
- OSCOAP: new App for Contiki
- Implementation based on v-04, with some differences:
  - No protected Observe option
  - No sliding window for sequence numbers
- Dependencies: COSE-C implementation (that uses CN-CBOR and adapted to use mbedTLS AES-CCM-64-64-128)
  - COSE-C is not optimized for embedded devices: large buffers are used. Some optimizations was done, to use one buffer only.
  - Crypto library
- The numbers reflect that

* http://people.inf.ethz.ch/mkovatsc/erbium.php

Message Overhead

AES-128-CCM-8 DTLS Record Layer Per-Packet Overhead with an 8-octet Integrity Check Value(ICV).

```
DTLS Record Layer Header................13 bytes
Nonce (Explicit).........................8 bytes
ICV..................................... 8 bytes
------------------------------------------------
Overhead................................29 bytes
------------------------------------------------
```

DTLS/DICE
draft-ietf-dice-profile

AES-128-CCM-8 OSCOAP Per-Packet Overhead with an 8-octet Integrity Check Value(ICV).

```
+----------+---------+----------+------------+
|   Tid    |   Tag   | COSE OH  | Message OH |
+----------+---------+----------+------------+
| 5 bytes  | 8 bytes |  9 bytes |  22 bytes  |
+----------+---------+----------+------------+
```

OSCOAP

- NOTE: This is NOT the minimum size of Transaction Identifier (Tid)

Network Overhead

› For the sum of Request + Response

	Set up connection (byte)	Data update (byte)	Close connection (byte)
Setup 1: CoAP using Non-confirmable messages	0	115	0
Setup 2: CoAP using Confirmable messages	0	115	0
Setup 3: CoAP with DTLS	2497	222	103
Setup 4: CoAP with OSCOAP	0	154	0

Figure 5.4: Measured network usage – test case 1: Single update

NOTE: DTLS handshake included but EDHOC is not

Execution Time

› OSCOAP not optimized: uses heap allocation functions, while DTLS uses block allocator

Table 7.3: Execution time of selected functions.

	Parse	Serialize	Decrypt	Encrypt
CoAP	0.0427 ms	0.0412 ms	N/A	N/A
OSCoAP	2.028 ms	2.668 ms	1.123 ms	1.327 ms
DTLS	0.836 ms	1.285 ms	0.8102 ms	0.8716 ms

Memory Footprint

› OSCOAP uses not optimized COSE-C implementation

Table 7.5: Memory footprint in bytes for a server using CoAP, OSCoAP and CoAP over DTLS

Application:	Server		
Protocol:	CoAP	OSCoAP	CoAP + DTLS
.bss	13799	14031	14899
.data	1772	2788	1922
.text	47070	77895	74498
RAM	15571	16819	16821
Flash	48842	80683	76420

Table 7.7: Code size for different parts of OSCoAP

	COSE-C	cn-cbor	mbedTLS	dynamic memory
Code size:	4770 Bytes	1394 Bytes	3734 Bytes	1714 Bytes

Flash and RAM

	Flash	RAM
Contiki	31.4	4.9
CoAP	8.4	0.6
DTLS - excl. crypto	 10.4	 1.0
OSCOAP -00	7.0	0.6
OSCOAP -04 - excl. crypto/COSE - COSE	 ~ 4 ~ 4.8	 < 1 < 1

Raza et al. "Lightweight Secure CoAP for theInternet of Things" (IEEE 2013)

- COSE and Crypto library of OSCOAP implementation not optimized
- CoAP and COSE can be reused for EDHOC

Improving the implementation

- Dynamic Memory Usage
  - C standard-library for dynamic memory on embedded systems is a bad idea
  - the code needed to manage the dynamic memory is larger then the amount of memory dynamically allocated.

- Unsuitable libraries
  - COSE and CBOR libraries contains code that is not used at all in the OSCoAP implementation.
  - This code duplicates a great deal of code from Erbium CoAP.

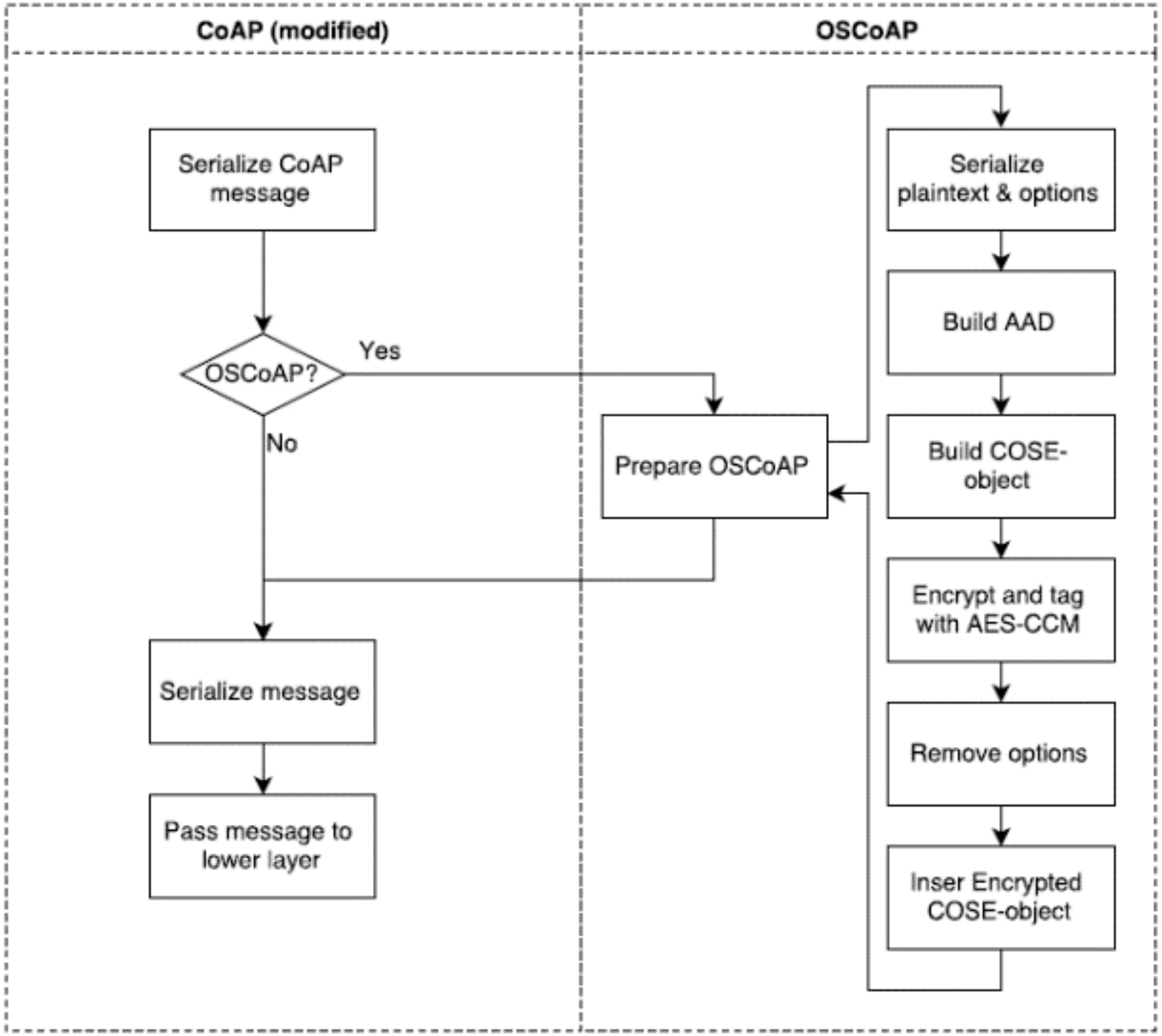
Thank you!

Comments/questions?

Backup slides

Implementation Details

Serialize Function



Implementation Details

Parse Function