“EDHOC is ~~designed~~implemented for highly constrained settings”

F. Molina, T. Claeys, E. Baccelli, M. Vučinić

IETF-113, LAKE
21 March 2022, 1330 UTC

Outline

1. Context & Use Case
2. Dependencies
    a. Available libraries
    b. Missing blocks
3. EDHOC-C: Some Benchmarks
4. Lessons Learned

Motivation

1. Context & Use Case

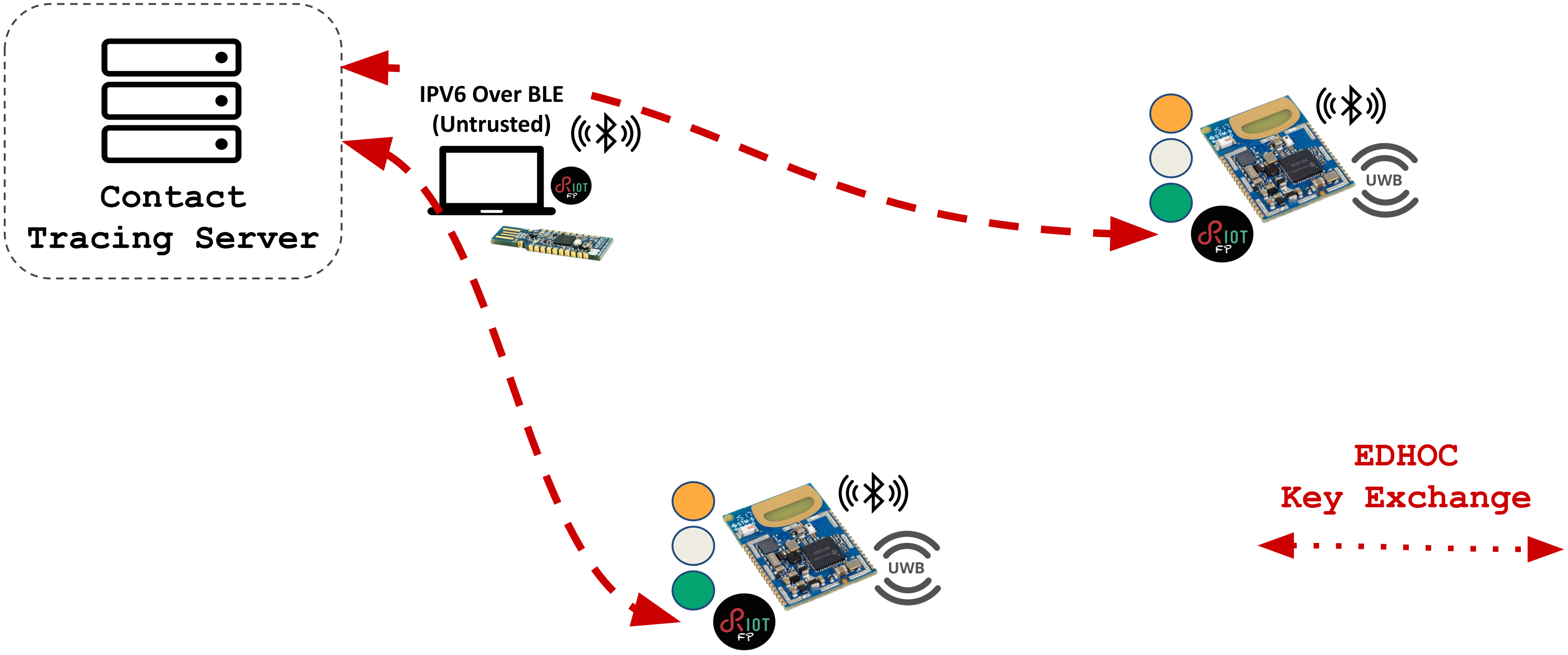
RIOT-fp: cybersecurity research project by Inria

- Developing high-speed, high-security, low-memory IoT crypto primitives
- Secure IoT software updates and supply-chain, over low-power networks
- Providing guarantees for software execution on low-end IoT

- More info on the RIOT-fp project website https://future-proof-iot.github.io/RIOT-fp/about

Use Case

1. Context & Use Case



Goals

1. Context & Use Case

- Generic C implementation of EDHOC for **microcontrollers**
  - Support all authentication methods
  - Support cipher-suites 1-4 (both ECDSA and Ed25519 signatures)
  - **Do not rely on hardware acceleration**
  - Optimized for embedded: no heap -> no malloc

- Reuse existing libraries (e.g., for crypto backend)
  - Reuse libraries that are likely to be used by our applications

- **Demonstrate integration in a constrained embedded software platform**
  - Running code on a large variety of microcontrollers!

Building blocks for the implementation

2. Dependencies

Spec base: **draft-ietf-lake-edhoc-05**

What do we need?

1. CBOR
   a. encoding / decoding
2. CRYPTO
   a. Key derivation
   b. Encryption/Decryption
   c. Signing/Verifying
3. Interoperability testing infrastructure
   a. Test vectors
   b. Communication infra + interop peer
   c. Interoperability “peer”
4. Embedded software platform/ecosystem (to integrate into)

Generic CBOR Library

2. Dependencies - Available

NanoCBOR

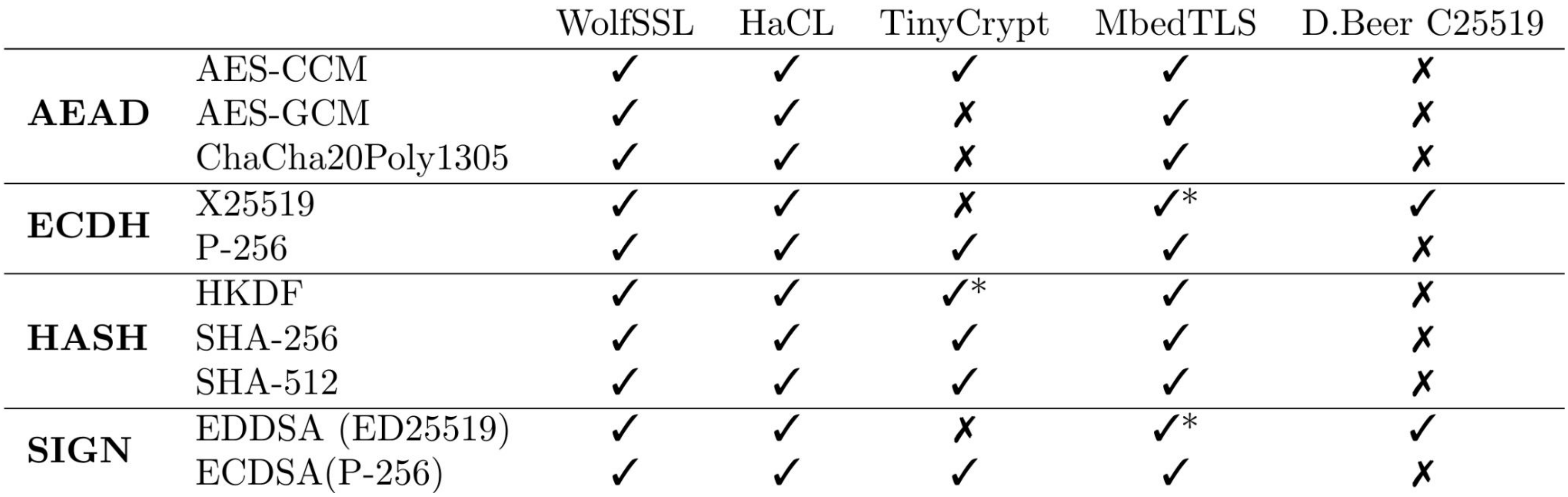
- ✅ Optimized for 32 bit and small footprint
- ✅ Decode -> check result
- ✅ No allocation
- ✅ Returns pointer to CBOR byte strings
- ❌ No functions for streaming CBOR
- ❌ Missing functions for easy map parsing

TinyCBOR

- ✅ Optimized for small footprint and fast execution
- ❌ Check type -> decode -> check result
- ✅ No allocation
- ❌ Copies content from CBOR byte strings

Cryptographic Backends (non-exhaustive)

2. Dependencies - Available



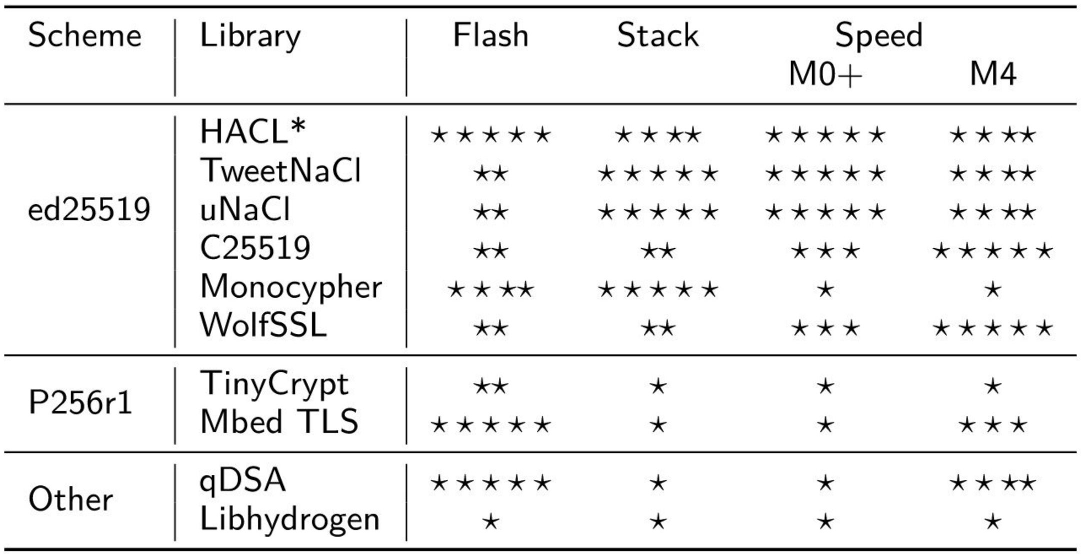
		WolfSSL	HaCL	TinyCrypt	MbedTLS	D.Beer C25519
AEAD	AES-CCM	✓	✓	✓	✓	✗
	AES-GCM	✓	✓	✗	✓	✗
	ChaCha20Poly1305	✓	✓	✗	✓	✗
ECDH	X25519	✓	✓	✗	✓*	✓
	P-256	✓	✓	✓	✓	✗
HASH	HKDF	✓	✓	✓*	✓	✗
	SHA-256	✓	✓	✓	✓	✗
	SHA-512	✓	✓	✓	✓	✗
SIGN	EDDSA (ED25519)	✓	✓	✗	✓*	✓
	ECDSA(P-256)	✓	✓	✓	✓	✗

✓* *Available in PRs or forked versions*

Cryptographic Backends (non-exhaustive)

2. Dependencies - Available

TABLE 2. Crypto library performance summary (fewer stars ⋆ is better).



Scheme	Library	Flash	Stack	Speed	
				M0+	M4
ed25519	HACL*	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆
	TweetNaCl	⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆
	uNaCl	⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆
	C25519	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
	Monocypher	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆	⋆
	WolfSSL	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
P256r1	TinyCrypt	⋆⋆	⋆	⋆	⋆
	Mbed TLS	⋆⋆⋆⋆⋆	⋆	⋆	⋆⋆⋆
Other	qDSA	⋆⋆⋆⋆⋆	⋆	⋆	⋆⋆⋆⋆
	Libhydrogen	⋆	⋆	⋆	⋆

K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig and E. Baccelli, "**Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check**," in *IEEE Access*, vol. 7, pp. 71907-71920, 2019, doi: 10.1109/ACCESS.2019.2919760.

Embedded Software Platform/Ecosystem

2. Dependencies - Available

Various open source options: FreeRTOS, RIOT, mbedOS, Zephyr, myNewt, liteOS…

We chose **RIOT** as general-purpose platform, which bundles:

- Generic HW support (ARM, RISC-V, MSP430, AVR, etc.)
- Ecosystem of libs, including
  - Crypto:
    - ✅ TinyCrypt
    - ✅ WolfSSL
    - ✅ AEAD & Hashes
    - ❌ MbedTLS (added since then)
    - ❌ HaCL (only old version supported)
  - CBOR libraries:
    - ✅ NanoCBOR
    - ✅ TinyCBOR
  - Network stacks:
    - ✅ CoAP/UDP/6LoWPAN (and 6TiSCH OpenWSN)
    - ✅ BLE, 802.15.4

COSE, Test Vectors & Interop

2. Dependencies - Missing

LibCoSE

A COSE abstraction of crypto libraries

- Backends
  - ✅ MbedTLS
  - ✅ HaCL
  - ❌ TinyCrypt (added since then)
  - ❌ WolfSSL
  - ✅ Monocypher
- ✅ Signatures
- ❌ Encrypt (no AES-CCM at the time, added since then)
- ✅ Stream based API
- ✅ No Malloc

❌ No Fully Supported Cipher Suite

❌ Direct Access to crypto still needed

Test Vectors

- ❌✅ Limited
  - No CBOR certificates
  - Not all methods
  - No real certificates

Interop

- ❌ Nothing at the time

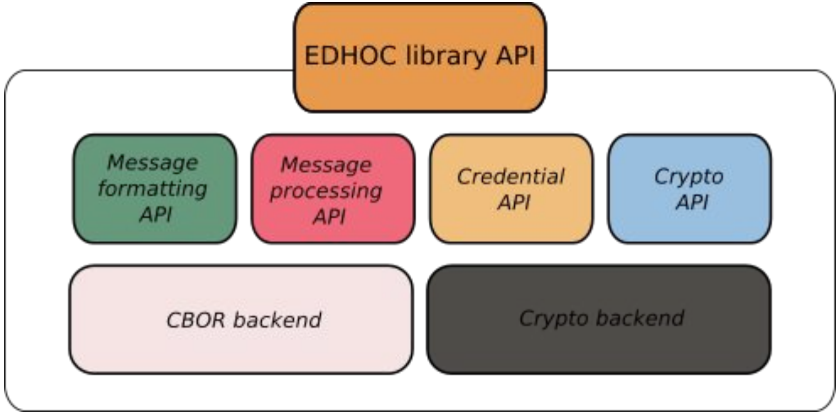
EDHOC-C

3. EDHOC-C: Some Benchmarks

Spec base: **draft-ietf-lake-edhoc-05**

What we used

1. CBOR
   a. **NanoCBOR**
2. CRYPTO
   a. Tincrypt: **AEAD & HASH**
   b. C25519 (D.Beer): **SIGN/VERIF & ECDH**
3. Interoperability testing infrastructure
   a. **py-edhoc**
   b. **CoAP**
4. Embedded software platform/ecosystem (to integrate into)
   a. **RIOT**



Package: https://github.com/openwsn-berkeley/EDHOC-C
RIOT Integration: https://doc.riot-os.org/group__pkg__edhoc__c.html

EDHOC-C Footprint RAM/ROM

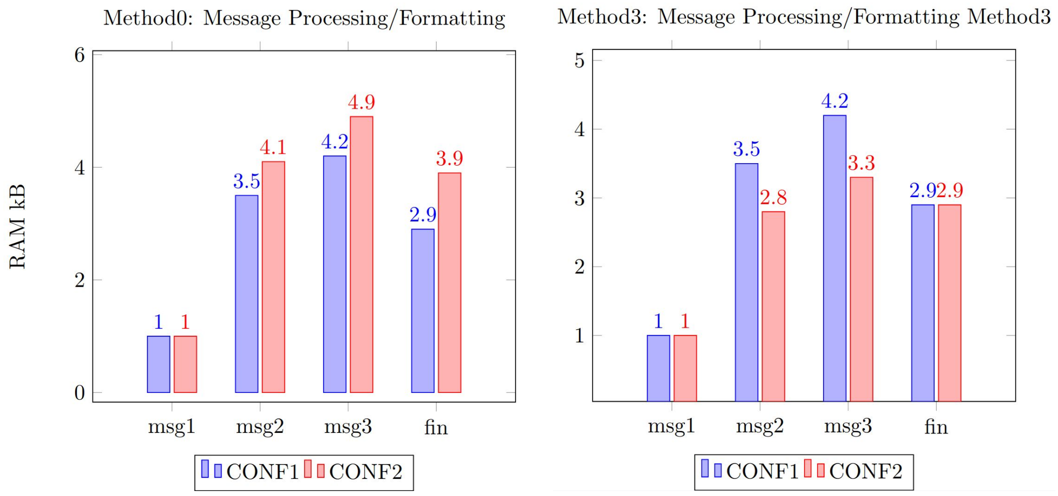
3. EDHOC-C: Some Benchmarks

`Cipher-suite-0, cortex-m4:`

➔ ROM: ~9kB

➔ RAM: highly dependent on:

- ◆ Method
- ◆ Additional Data Size
- ◆ Credentials Size
- ◆ Credentials ID Size

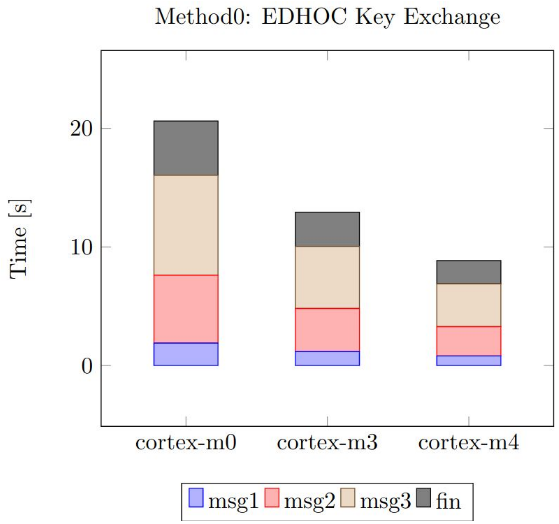


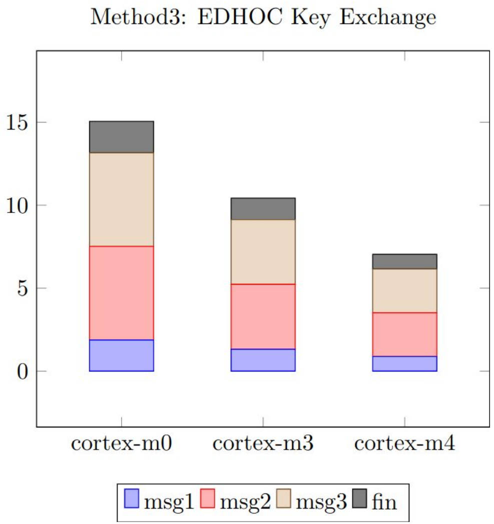
RAM usage configurations

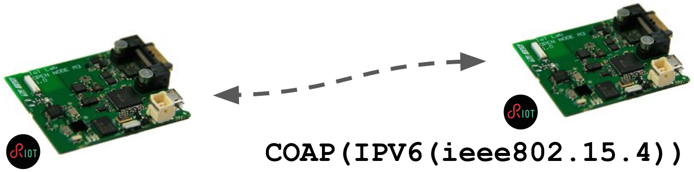
	Additional Data	Credentials	Credentials ID
CONF1	64B	256B	256B
CONF2	0B	128B	1B

EDHOC-C Handshake (`cipher-suite-0`)

3. EDHOC-C: Some Benchmarks (No HW Acceleration)







COAP(IPV6(ieee802.15.4))
(over BLE: similar results)

EDHOC-C Handshake (`cipher-suite-0`)

3. EDHOC-C: Some Benchmarks (No HW Acceleration)

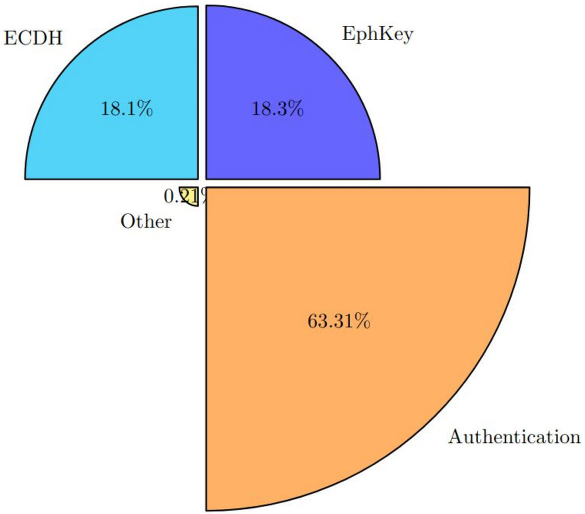


Figure 1: Method0: EDHOC Key Exchange

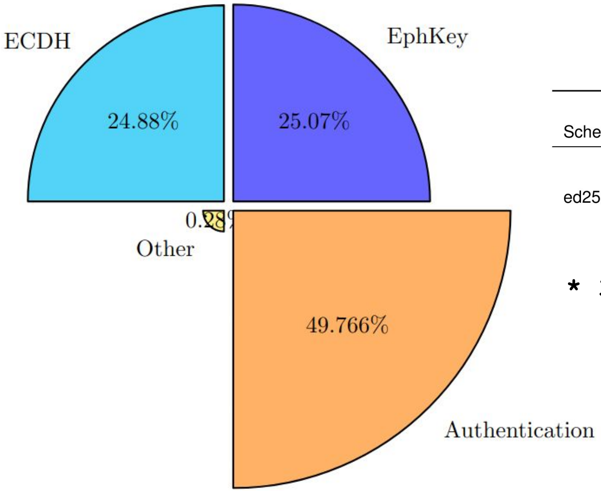


Figure 2: Method3: EDHOC Key Exchange

		Cortex M0+		Cortex M3		Cortex M4	
Scheme	Library	sign	verify	sign	verify	sign	verify
ed25519	HACL*	6830	7067	1480	1526	1238	1279
	TweetNaCl	3998	7983	998	1988	730	1452
	uNaCl	4050	8086	906	1804	751	1495
	C25519	1787	4178	1450	3326	838	1938
	Monocypher	189	529	38	72	25	45
	WolfSSL	1770	3652	1331	2686	829	1698

`* 32 bytes message`

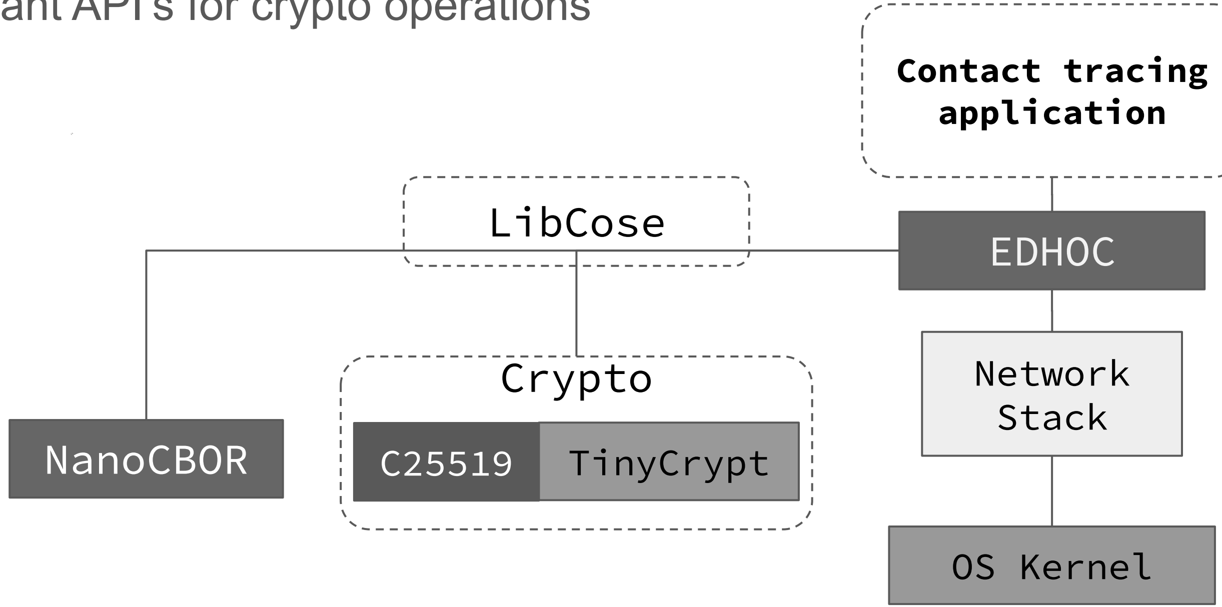
*** DISCLAIMER: coarse measurements.**

K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig and E. Baccelli, "**Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check**," in *IEEE Access*, vol. 7, pp. 71907-71920, 2019, doi: 10.1109/ACCESS.2019.2919760.

EDHOC-C on RIOT: test in the field

4. Lessons Learned

1. Application uses LibCOSE
    a. conflicts and redundant with EDHOC-C own COSE
2. Re-entrant API's for crypto operations



EDHOC-C (draft-05 implementation)

4. Lessons Learned

1. Who and how to parse credentials?
2. `bstr_identifier` savings (1 byte) not worth the extra code complexity
3. Optimizing Ram
   a. cose-key structures allocating space for the x,y,d,sym
   b. Limit credentials ID support (no full credential)
   c. Statically allocated work buffers: msg_struct, cose_keys, key_streams, etc..
      i. tricky to know when to assume everything is allocated on the stack or not
4. Used cipher-suite-0 (only one available), **SHOULD use `cipher-suite-2/3`**
   a. Re-use BLE crypto requirements
   b. No sha512 required
5. **MIGHT use `cipher-suite-5`** for code size (if no AES-CCM already)
   a. AES-CCM no implementation with incremental API
   b. ChaCha20-Poly1305 code size smaller than AES-CCM (in RIOT)
   c. No sha512 required