“EDHOC is designed implemented for highly constrained settings”

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

Contact Tracing Server

IPV6 Over BLE (Untrusted)

EDHOC Key Exchange

4
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

<table>
<thead>
<tr>
<th>NanoCBOR</th>
<th>TinyCBOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Optimized for 32 bit and small footprint</td>
<td>✔ Optimized for small footprint and fast</td>
</tr>
<tr>
<td>✔ Decode -&gt; check result</td>
<td>execution</td>
</tr>
<tr>
<td>✔ No allocation</td>
<td>✔ No allocation</td>
</tr>
<tr>
<td>✔ Returns pointer to CBOR byte strings</td>
<td>✔ No allocation</td>
</tr>
<tr>
<td>✗ No functions for streaming CBOR</td>
<td>✗ Check type -&gt; decode -&gt; check result</td>
</tr>
<tr>
<td>✗ Missing functions for easy map parsing</td>
<td>✗ Copies content from CBOR byte strings</td>
</tr>
</tbody>
</table>
## Cryptographic Backends (non-exhaustive)

### 2. Dependencies - Available

<table>
<thead>
<tr>
<th></th>
<th>WolfSSL</th>
<th>HaCL</th>
<th>TinyCrypt</th>
<th>MbedTLS</th>
<th>D.Bea</th>
<th>C25519</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AEAD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-CCM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>AES-GCM</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>ChaCha20Poly1305</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>ECDH</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X25519</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓*</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>P-256</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>HASH</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HKDF</td>
<td>✓</td>
<td>✓</td>
<td>✓*</td>
<td></td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>SIGN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDDSA (ED25519)</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓*</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ECDSA (P-256)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

✓* Available in PRs or forked versions
Cryptographic Backends (non-exhaustive)

2. Dependencies - Available

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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Library</th>
<th>Flash ★★★★★</th>
<th>Stack ★★★★★</th>
<th>Speed M0+ ★★★★★</th>
<th>M4 ★★★★★</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed25519</td>
<td>HACL*</td>
<td>★★★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>TweetNaCl</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>uNaCl</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>C25519</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>Monocypher</td>
<td>★★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>WolfSSL</td>
<td>★★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>P256r1</td>
<td>TinyCrypt</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>Mbed TLS</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Other</td>
<td>qDSA</td>
<td>★★★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td></td>
<td>Libhydrogen</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
</tr>
</tbody>
</table>

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

**Test Vectors**

- ✗ Limited
  - ✗ No CBOR certificates
  - ✗ Not all methods
  - ✗ No real certificates

**Interop**

- ✗ Nothing at the time

- ✗ No Fully Supported Cipher Suite
- ✗ Direct Access to crypto still needed
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

<table>
<thead>
<tr>
<th>RAM usage configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Additional Data</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>CONF1</td>
</tr>
<tr>
<td>CONF2</td>
</tr>
</tbody>
</table>
EDHOC-C Handshake (cipher-suite-0)

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

COAP(IPV6(ieee802.15.4))
(over BLE: similar results)
3. EDHOC-C: Some Benchmarks (No HW Acceleration)

* DISCLAIMER: coarse measurements.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Library</th>
<th>Cortex M0+</th>
<th>Cortex M3</th>
<th>Cortex M4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sign</td>
<td>verify</td>
<td>sign</td>
<td>verify</td>
</tr>
<tr>
<td>HACL*</td>
<td>6830</td>
<td>7067</td>
<td>1480</td>
<td>1526</td>
</tr>
<tr>
<td>TweetNaCl</td>
<td>3998</td>
<td>7983</td>
<td>998</td>
<td>1988</td>
</tr>
<tr>
<td>uNaCI</td>
<td>4050</td>
<td>8086</td>
<td>906</td>
<td>1804</td>
</tr>
<tr>
<td>C25519</td>
<td>1787</td>
<td>4178</td>
<td>1450</td>
<td>3326</td>
</tr>
<tr>
<td>Monocypher</td>
<td>189</td>
<td>529</td>
<td>38</td>
<td>72</td>
</tr>
<tr>
<td>WolfSSL</td>
<td>1779</td>
<td>3652</td>
<td>1331</td>
<td>2686</td>
</tr>
</tbody>
</table>

* 32 bytes message
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