www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable n°2.3 : Benchmarking Results of a Zero-Touch Lifecycle Framework

Work Package n°2: Orchestration of Collaborative Smart Nodes

Task Lead: INRIA

WP Lead: ADI

This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101093046.

Funded by the European Union

Document information	
Author(s)	Mališa Vučinić, Geovane Fedrecheski
Reviewers	Dara Osullivan, Thomas Watteyne
Submission date	31-Oct-2024
Due date	31-Oct-2024
Type	Report
Dissemination level	Public

Document history

Date	Version	Author(s)	Comments
25-Jun-2024	01	Mališa Vučinić	preview
31-Oct-2024	02	Mališa Vučinić, Geovane Fedrecheski	deliverable

Table of contents

Executive Summary

This document details the results achieved in the context of Task 2.2 and specifically the development and benchmarking of the zero-touch network enrollment solution for embedded devices. The effort within this Task focused on standardizing the zero-touch solution within the Internet Engineering Task Force (IETF) standardization body and providing scientific studies on its performance results and comparative advantages to the existing solutions available. The zero-touch enrollment protocol operates over a new security handshake standard in the IETF called the EDHOC security handshake protocol, formalized in RFC 9528 and RFC 9529. The deliverable gathers the content regarding the development of EDHOC, EDHOC's comparative performance study with DTLS 1.3, the detailed technical specification of the zero-touch enrollment solution operating over EDHOC and finally the benchmarking results of the zero-touch enrollment solution. The results of this Deliverable enable OpenSwarm devices to securely join a swarm without any swarm-deployment-specific information provisioned to the devices. This means that a swarm device can join a swarm out-of-the-box as it was received from its manufacturer, with the intelligence shifted to the link between the manufacturer (online entity) and the swarm operator. Additionally, the solution outlined in this document can be used for secure joining of ADI's SmartMesh IP network nodes (T2.1).

Introduction

Enrolling a new device is a fundamental procedure that allows systems to grow from N registered devices to N + 1. In the Internet of Things (IoT), enrollment is commonly performed when devices are deployed, where network connection and trust must be established for the first time. It requires strong security guarantees to ensure that a newly joined device is not a malicious actor, which could lead to unauthorized use of in-network resources and leakage of sensitive data. In addition, given the constrained nature of IoT devices and networks, it must be optimized in terms of computing and network usage. In fact, to avoid Denial of Service (DoS) attacks, some network technologies restrict even more the data rate during network formation [1], further stressing the need for a lightweight enrollment method.

Another important requirement is the operational overhead during deployment. In cases where large deployments are made at once, such as in industrial IoT or in robotic swarms [2], the cost of configuring each device with the right credentials, directly on the field, may be prohibitive. Furthermore, manual configuration implies in giving operational staff access to sensitive credentials, thus exposing another attack vector. The ideal solution would allow devices to just turn on at the deployment location and connect to the infrastructure without requiring user configuration, enabling what we call “zero-touch” enrollment.

Every IoT communication technology needs a way of enrolling new devices. From long-range communication protocols such as LoRaWAN [3] to short-range and low-power such as 6TiSCH [1] and Bluetooth Low Energy (BLE) [4], all of them provide a native enrollment mechanism. To address common limitations in technology-specific enrollment mechanisms, such as the use of pre-shared keys or the need for user interaction, some technologies integrate more flexible authentication methods. This is the case of Wireless Fidelity (Wi-Fi) [5], which leverages the Extensible Authentication Protocol (EAP) to facilitate network join, making it a widely used solution in enterprise and academic networks that require both scalability and higher security levels [6].

The design of security protocols for IoT environments must take into account requirements to minimize the use of resources. To address this gap, the Internet Engineering Task Force (IETF) has been developing new protocols that provide high levels of protection, while optimizing metrics such as message sizes and count, CPU and memory usage, as well as energy consumption and execution time. For example, Object Security for Constrained RESTful Environments (OSCORE) [7] and Ephemeral Diffie-Hellman Over COSE (EDHOC) [8] were developed to address, respectively, needs of end-to-end message security and authenticated key exchange with minimal overhead.

The work in the context of T2.2 is therefore positioned around the activities in the IETF. The different pieces that allow a network-agnostic, lightweight and zero-touch enrollment are standardized or currently under standardization and are presented in this Deliverable. The OpenSwarm DoA envisioned performing the benchmarking results of this Deliverable on the OpenTestbed, an in-house Inria Paris testbed based on OpenMote-CC2538 devices. The OpenTestbed has in the meantime been discontinued by Inria. Therefore, we performed the benchmarking on an in-house testbed consisting of OpenSwarm devices, as detailed further in this Deliverable.

The rest of this Deliverable is organized as follows. First, we overview the OpenSwarm Key Performance Indicators planned and those achieved, the reached Technology Readiness Level (TRL) and the overview of the publication resulting from the work on T2.2. Then, we introduce the standardization process within the IETF and how OpenSwarm contributed to it. We describe the State of the Art with respect to the different enrollment techniques for the Internet of Things and beyond. We then describe the base protocol of our solution, whose standardization OpenSwarm contributed to, the EDHOC protocol, as well as the lightweight data encryption layer called OSCORE. We describe in detail the specification of the zero-touch extension to EDHOC which we term as "**ELA: EDHOC with Lightweight Authorization".** We first benchmark the EDHOC protocol against the Datagram Transport Layer Security 1.3 handshake protocol and then focus on ELA compared to the most relevant EAP-based enrollment solution. We summarize our contributions and the work done in the Conclusion.

Key Performance Indicators (KPIs)

The Key Performance Indicator (KPI) defined for Task 2.2 is shown in Table 1. It covers the number of standardization contributions, including IETF Internet Drafts, of the task. For the duration of the Task, we have published one RFC (RFC 9529) and one adopted IETF Internet Draft (draft-ietf-lake-authz-01) specifying the zero-touch solution. The latter has become an official document of the IETF, meaning that it is on a well-defined path to becoming an RFC, as explained in Section Standardization. We have also published another standardization contribution, in the context of Task T2.3 on remote attestation (Internet Draft draft-song-lake-ra) but do not consider it in this section as it is relevant to T2.3. Therefore, within the direct scope of Task 2.2, we have published a total of **2** standardization contributions to the IETF.

Table 1. KPI for Task T2.2

Task	Key Performance Indicator	Target	Achieved
T2.2	Number of standardization contributions (including IETF Internet Drafts)	At least 2	2

Technology Readiness Level (TRL)

The solution developed in the context of Task 2.2 is under standardization within the Internet Engineering Task Force, as explained in the Section Standardization. Before the start of the OpenSwarm project, the zero-touch solution has been formulated but no implementations existed (**TRL 2**)**.** Inria has delivered an implementation of this zero-touch joining solution and it is available under an open-source 3-clause BSD license as part of the lakers library [1], gathering protocols standardized within the IETF LAKE working group. Inria has demonstrated the developed zero-touch technology in the lab and integrated it with the DotBot platform, video being available on YouTube [2]. We conclude that the TRL of the zero-touch solution is currently **TRL 4**.

Table 2. Technology readiness level of the zero-touch joining solution.

Technology Developed	Start TRL	Current TRL
Zero-touch joining solution for DotBots	TRL 2	TRL 4

[1] https://github.com/openwsn-berkeley/lakers

[2] https://www.youtube.com/watch?v=0kAtbMMcVRg

List of Publications

- Secure Communication for the IoT: EDHOC and (Group) OSCORE Protocols. Rikard Hoglund, Marco Tiloca, Goran Selander, John Preuss Mattsson, Mališa Vučinić, Thomas Watteyne. IEEE Access, 2024.
- Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments. Geovane Fedrecheski, Mališa Vučinić, Thomas Watteyne. IEEE Wireless Communications and Networking Conference (WCNC), Dubai, United Arab Emirates, 21-24 April 2024.
- Traces of Ephemeral Diffie-Hellman over COSE (EDHOC). Goran Selander, John Preuss Mattsson, Marek Serafin, Marco Tiloca, Mališa Vučinić. Internet Engineering Task Force, RFC 9529.
- Lightweight Authorization using Ephemeral Diffie-Hellman over COSE. Goran Selander, John Preuss Mattsson, Mališa Vučinić, Geovane Fedrecheski, Michael Richardson. Internet Engineering Task Force, draft-ietf-lake-authz-01.

Standardization

In this section, we detail the standardization process we followed to standardize the solution outlined in this Deliverable, how the Internet Engineering Task Force (IETF) works, the scope of work in the Lightweight Authenticated Key Exchange (LAKE) working group, and specifically the OpenSwarm contributions to that process.

Internet Engineering Task Force (IETF) is the open standardization body behind most of today's Internet. It was founded in 1986 and publishes voluntary standards that are "often adopted by Internet users, network operators, and equipment vendors". The participation to IETF activities is open and there is no such thing as formal membership.

Most of the work happens on public mailing lists. The IETF currently splits its work into 127 working group, divided into 7 areas.

For a document to become an RFC (Request for Comments), a type of an Internet Standard, it typically undergoes 3 stages. First, it is submitted as an individual submission, carrying the name of the form *draft-lastname-workinggroupname-docname*. Second, after consideration and lengthy discussions and reviews, some documents are *adopted* by the appropriate working groups. In this stage, the document becomes an official document of the IETF, and bears the name of the form *draft-ietf-workinggroupname-docname*. The authors of the document in this stage incorporate the feedback of the working group and continuously work on publishing new revisions of the document. Third, once the working group declares the document as ready, the document undergoes a series of reviews from the IETF experts outside of the working group (Area Director, Internet Engineering Steering Group) and is finally approved to be published as an RFC. The process is open, and anyone can at any point raise issues on the technical contents of the document. The time it takes for a document to undergo the different stages and become an RFC depends on many factors but is typically not less than 2 years.

OpenSwarm contributions are within the Security Area and specifically within the Lightweight Authenticated Key Exchange (LAKE) working group. The LAKE working group is co-chaired by Inria (Dr. Mališa Vučinić). The LAKE working group standardized in early 2024 a new security handshake protocol called EDHOC and is now working on standardizing the extensions to the EDHOC protocol. One of those extensions is the zero-touch joining protocol operating over EDHOC. Another extension is the procedure for remote attestation, part of OpenSwarm Task 2.3 (detailed in Deliverable D2.4).

In May 2024, OpenSwarm project organized a Hackathon gathering the IETF community interested in "Lightweight IoT Security". The event held at the premises of Inria Paris gathered approximately 40 participants and was used as an opportunity to progress the documents related to Deliverables D2.3 (this deliverable) and D2.4, in the scope of Tasks 2.2 and 2.3. We also used the opportunity to present demos of ongoing work in the scope of OpenSwarm, thus contributing to dissemination activities. More information about the Hackathon organized by OpenSwarm is available at the official Hackathon website [3] and a video is available on YouTube [4].

Figure 1. Participants of the Hackathon on Lightweight IoT Security gathering the IETF community, held at Inria Paris on 21-22 May 2024.

State of the Art

There are several existing protocols for authorization and enrollment of new IoT devices. We first describe related works whose join procedures are specific to the underlying network, and then describe those that are flexible with respect to the lower layer.

3 https://parishackathon.lakewg.org

4 https://www.youtube.com/watch?v=s_fC_CgtHvY

LoRaWAN is a widely used Low-Power Wide-Area Network (LPWAN) technology, featuring a maximum data rate of 27 kb/s and acting in the kilometer range. It supports an over the air activation procedure where a pre-shared key (AppKey) is used as a root key from which session keys are subsequently derived. During the join procedure, two small messages (< 30 bytes each) are exchanged between device and gateway. Among its limitations, the procedure is technology-specific, reveals the device identifier (DevEUI), and uses pre-shared keys which if compromised may expose communications of the whole network.

IEEE 802.15.4 [9] defines the physical and MAC layers for low-rate networks (up to 250 kb/s) between devices [1] up to tens of meters indoors, hundreds of meters outdoors. Several higher layer protocols leverage IEEE 802.15.4 to offer solutions with different trade-offs, each with its own security setup procedure.

For example, ZigBee, a protocol based on IEEE 802.15.4 that offers higher layer solutions for creating personal area networks [10], uses a solution comparable to LoRaWAN, in which a pre-shared master key is used to bootstrap a network key that is used during the join process.

The IETF 6TiSCH protocol stack [1] building on top of time-synchronized channel hopping (TSCH) mode of IEEE 802.15.4 also leverages a pre-shared key for network join. This symmetric key is a shared between the joining node and a central entity. The key is different for each node in the network [11] [12]. The join protocol consists of two messages Join Request and if successful, Join Response, protected with Object Security for Constrained RESTful Environments (OSCORE). While efficient in terms of over-the-air overhead, the protocol relies on a provisioned pre-shared key on each device, which requires a "touch" of each device during the deployment.

BLE enables connections of up to 2 mb/s between devices with a range similar to that of IEEE 802.15.4. It supports several levels of security [12] [13], from no security at all up to authenticated key exchange with asymmetric keys and man- in-the-middle (MITM) protection. Protection for MITM is achieved by employing out-of-band user-assisted interaction; devices with limited input/output sources can skip such protection. Secure associations can be established via a pairing procedure that involves negotiation of security and IO capabilities. Limitations of BLE include the need for user interaction to achieve higher levels of security, which limits its scalability in large deployments.

Wi-Fi allows high-speed connections at the cost of higher power consumption. Wi-Fi allows nodes joining without security, with the use of pre-shared keys, or based on port-based access control, such as IEEE 802.1X [13]. Since IoT devices often lack user interfaces, Wi-Fi Alliance has also developed the Device Provisioning Protocol (DPP), which allows user-assisted enrollment of IoT devices via a configurator device (e.g. a smartphone). In DPP, a configurator device uses an out-of-band channel (e.g. a QR code) to establish a secure connection with the device, through which the details of the target Wi-Fi network can be sent, e.g., SSID and password. Similarly to BLE, a significant limitation of DPP is that the user needs to interact with each new device being enrolled in the network.

A different class of protocols implements authorization in a way that is independent of the underlying transport. This is the case of EAP, an authentication framework that supports multiple authentication methods [14]. EAP defines the base protocol messages and handling of duplicates and retransmission but leaves the specifics of authentication to companion specifications. For example, EAP-TLS [15] defines an authentication method using Transport Layer Security, enabling entities to authenticate using public key certificates. EAP defines the authentication to occur between a peer and an authenticator, but also considers the case where the latter acts like a pass-through gateway, forwarding the EAP packets to an authentication server. This allows flexibility to perform either local or centralized authorization management, depending on the application needs.

Variants of EAP have been developed to tackle specific challenges found in IoT networks. The EAP-NOOB (Nimble Out-Of-Band) method [16] allows bootstrapping IoT devices that have no pre-provisioned credentials. It performs a user-assisted procedure to exchange messages via a non-network out-of-band channel, such as display or speaker. In the end, the EAP peer is provisioned with the credentials needed to join a particular network. Like most BLE methods and DPP, this approach has the drawback of requiring per-device configuration during deploy, limiting its scalability.

The EAP-EDHOC method [17] specifies how to authenticate EAP-enabled entities using EDHOC. EDHOC is a very compact key exchange protocol with mutual authentication, identity protection, and forward secrecy. A main goal of EDHOC is to be used in scenarios where computing resources are scarce, such as the IoT. The EAP-EDHOC method enables integration of EDHOC-based authentication in different applications and use cases using the EAP framework. Although this method largely benefits from EDHOC's compactness, it still has to exchange a number of extra messages that are needed by EAP to begin and finish the protocol.

EAP is also directly used by IEEE 802.1X, a port-based network access control protocol that authenticates devices trying to join a network [13]. IEEE 802.1X also defines EAP Over LAN (EAPOL), an encapsulation format for carrying EAP packets in wired and wireless networks, such as Ethernet and Wi-Fi. IEEE 802.1X follows an architecture that is compatible with EAP, allowing a supplicant (the EAP peer) to join a network via an authenticator, which in turn may implement authorization capabilities or delegate it to a dedicated server. This server typically implements the Remote Authentication Dial In User Service (RADIUS) protocol, which is used to carry authentication and authorization messages exchanged with the authenticator. A limitation of IEEE 802.1X is that it was developed for non-constrained networks, such as Wi-Fi, and therefore is not directly applicable to IoT networks.

Eduroam [6] is a widely deployed network access service that simplifies network access for users in academic and research institutions. It utilizes IEEE 802.1X and EAP, specifically relying on authentication methods like EAP-TLS for secure, federated identity verification. Through eduroam, a device can authenticate with its home institution's credentials while accessing a network provided by a participating institution, enabling seamless roaming across networks. The authentication occurs through a hierarchical structure of RADIUS servers, which forward the credentials securely to the user's home institution for validation. This method leverages the EAP framework to ensure flexibility in transport protocols and authentication mechanisms, making it scalable and suitable for large academic and research networks. Similarly to IEEE 802.1X, eduroam is not applicable for constrained networks.

Bootstrapping Remote Secure Key Infrastructure (BRSKI) [18] is a protocol designed to securely onboard IoT devices into a network by automating the process of enrollment and provisioning. BRSKI extends existing Public Key Infrastructure (PKI) mechanisms and builds upon the IETF's Enrollment over Secure Transport (EST) protocol. It leverages certificate-based authentication for devices, ensuring mutual authentication between devices and the network. During the join procedure, BRSKI uses a voucher exchange system between the device, a registrar, and a manufacturer-authorized server to authenticate the device's identity and provide it with the necessary credentials to access the network securely. However, BRSKI was not designed to work with constrained environments, and a typical voucher exchange is in the order of kilobytes of data [18].

Taking inspiration from fully fledged network join solutions such as IEEE 802.1X and learning from ways to implement secure but lightweight device enrollment solutions for IoT use cases, in this paper we propose Lightweight Authorization using EDHOC. LAE is improves upon related work by enabling a very compact and low-message count authorization solution that is applicable for enrollment of constrained IoT devices, in a way that is agnostic to the underlying network. It allows obtaining a network access voucher with an overhead of less than 100 bytes, and secures communications in all ends: device, domain authenticator, and enrollment server. Finally, LAE benefits from the strong security guarantees provided by EDHOC and fills a gap in its specification by utilizing the extension point in EDHOC to carry the authorization-related information.

Secure Communication for the IoT: EDHOC and (Group) OSCORE Protocols

The following work was published as "Secure Communication for the IoT: EDHOC and (Group) OSCORE Protocols. Rikard Hoglund, Marco Tiloca, Goran Selander, John Preuss Mattsson, Mališa Vučinić, Thomas Watteyne. IEEE Access, 2024."

Summary

Communication security of an Internet-of-Things (IoT) product depends on the variety of protocols employed throughout its lifetime. The underlying low-power radio communication technologies impose constraints on maximum transmission units and data rates. Surpassing maximum transmission unit thresholds has an important effect on the efficiency of the solution: transmitting multiple fragments over low-power IoT radio technologies is often prohibitively expensive. Furthermore, IoT communication paradigms such as one-to-many require novel solutions to support the applications executing on constrained devices. Over the last decade, the Internet Engineering Task Force (IETF) has been working through its various Working Groups on defining lightweight protocols for Internet-of-Things use cases. "Lightweight" refers to the minimal processing overhead, memory footprint and number of bytes in the air, compared to the protocol counterparts used for non-constrained devices in the Internet. This section overviews the standardization efforts in the IETF on lightweight communication security protocols. It introduces EDHOC, a key exchange protocol, as well as OSCORE and Group OSCORE, application data protection protocols adapted for securing IoT applications. The article additionally highlights the design considerations taken into account during the design of these protocols, an aspect not present in the standards documents. Finally, we present an evaluation of these protocols in terms of

the message sizes, and we compare them with the non-constrained counterpart, the (D)TLS protocol. We demonstrate that the novel key exchange protocol EDHOC achieves ×5 reduction over DTLS 1.3 authenticated with pre-shared keys in terms of total number of bytes transmitted over the air, while keeping the benefits of authentication with asymmetric credentials.

Building Blocks

The main building blocks for constrained low-power wireless systems are: CoAP, CBOR, COSE and C509. Both secure one-to-one communication and secure one-to-many group communication, detailed below, build on those.

CoAP: CONSTRAINED APPLICATION PROTOCOL

The Constrained Application Protocol (CoAP) standard [19] is a web transfer protocol adapted for constrained networking technologies and similar to HTTP.

CoAP especially supports IoT communication paradigms such as group communication and asynchronicity. The protocol is compact: a message starts with a fixed 4-byte header, possibly extended by options before the application payload. CoAP operates on a request-response model with optional response asynchronicity through the Observe extension. Group communication is supported through an extension to the core protocol, and relies on one-to-many requests (e.g., over IP multicast) and unicast responses. One core feature of CoAP is response caching at intermediaries. This allows a client to retrieve an IoT device's data even when that device has its radio off.

CBOR: CONCISE BINARY OBJECT REPRESENTATION

The Concise Binary Object Representation (CBOR) standard is a data format designed for encoding compactness, small code size and extensibility. It is based on the JavaScript Object Notation (JSON) data model but uses binary encoding. CBOR defines several "major types" such as unsigned and negative integers, byte and text strings, arrays, and dictionaries. CBOR is a basic building block of many IoT protocols standardized in the IETF, including the security protocols discussed in this article.

COSE: CBOR OBJECT SIGNING AND ENCRYPTION

The CBOR Object Signing and Encryption (COSE) standard describes how to create and process representations for cryptographic keys, ciphertexts, signatures, message authentication codes and key exchange, using CBOR for serialization. COSE builds on JSON Object Signing and Encryption, but, since it uses CBOR, COSE objects are smaller. COSE is designed for allowing constrained devices to verify protected messages efficiently.

C509 CERTIFICATES

CBOR-encoded X.509 (C509) Certificates is an ongoing IETF work [20] to standardize a CBOR encoding of X.509 certificates suitable especially for the IoT. The CBOR encoding already supports a large subset of X.509 certificates. The C509 encoding can, in many cases, reduce the size of certificates by over 50%, and can be used independently of certificate compression, which, when combined, may provide additional reduction. Two different types of C509 certificates are specified, and differ only in the content of the signature field: 1) for CBOR re-encoded X.509 certificates, the signature is made over the original DER- and ASN.1- encoded X.509 certificate; 2) for natively signed C509 certificates, the CBOR encoding itself is signed. The latter avoids the ASN.1 processing and the decoding process, both of which are complex for constrained devices. While natively signed C509 significantly simplifies the processing at constrained devices, it loses backwards compatibility with existing X.509 certificates until CBOR encoding is deployed in CAs, for which the CBOR re-encoded X.509 provides a migration path.

Secure One-to-One Communication

We start by describing one-to- one communication using EDHOC for key establishment and OSCORE for message protection.

Example Scenario

Without loss of generality, we refer to an industrial scenario to illustrate what the needs are and what the standards offer.

We consider the scenario of a factory floor at an industrial plastics manufacturer, where 20 machines on the floor produce plastic bottles. Each machine is equipped with a hopper in which plastic granulate gets poured every now and then. The machine melts these granulates, injects the molten plastic into a mold, and produces a plastic bottle which falls in a container every 30 s.

Previously, workers had to manually monitor the hopper state, pour granulates when its level was low, and monitor the container to swap it when full of bottles. Failing to keep the hopper full caused the machine to ''run empty'', and restarting it took a skilled technician 1-2 hours of work, during which the machine was down. Across three shifts, this happened 1-2 times per month.

Today, this is automated through each machine equipped with three sensors and two actuators. A level sensor measures the fill level of the hopper, and granulates can be poured into the hopper from a pipe using a solenoid valve. Then, a weight sensor monitors the fill level of the container of bottles, and an industrial blinking light can be switched on so that a worker can swap the container. On top of this, a wireless transceiver is installed on the extension port of the machine and publishes internal values (number of bottles produced, internal temperature, state of the motors). Each of these five devices is battery powered and communicates wirelessly to avoid complex cabling. The wireless medium exposes the operations to a variety of threats including denial-of-service, decreased production quality or quantity, safety of employees, ransomware and industrial espionage, from adversary agents within coverage.

With 20 machines on the factory floor, there are 100 devices in total. A wireless network interconnects all these devices with the gateway, itself connected to control software running on a server in the company's cloud. The control software receives and logs all sensor values, and triggers the opening of the valves and the switching of the lights, when needed. Given its criticality in the production system, communication between the wireless devices and the control software must be secure. Unfortunately, the Internet connectivity between the gateway on the factory floor and the control software may go down for short periods of time. Therefore, the gateway implements a simple proxy functionality by caching packets whenever the Internet connection is down.

The following sections describe the different protocols used for securely integrating devices in the factory network, and to be able to establish and maintain secure interactions among those. All protocols build on the building blocks discussed above.

EDHOC: Establishment Of Keying Material

The Ephemeral Diffie-Hellman Over COSE (EDHOC) [8] [21]is the protocol that establishes a shared symmetric key between two devices. EDHOC is based on an authenticated Diffie-Hellman key exchange. It is designed to be lightweight with low overhead, and suitable to resource-constrained environments.

To achieve mutual authentication, EDHOC uses peer authentication credentials, e.g., public-key certificates. To reduce the size of EDHOC messages, the authentication credentials can also be transported by reference, instead of by value. If an authentication credential is transported by reference, its value is typically distributed out-of-band and then retrieved from the local storage using the reference. Through this and other design decisions, EDHOC achieves a small message size which compares favorably to alternatives such as (D)TLS.

The overall security design of the protocol is based on the SIGMA-I MAC-then-SIGN variant [21] that provides: forward secrecy, the guarantee that a compromise of long-term authentication credentials or of a session key does not compromise past session keys; protection of identities; and peer mutual authentication. EDHOC uses COSE for cryptographic operations and CBOR for data encoding, and is typically transported over CoAP. The EDHOC design also provides negotiation and support for a range of cipher suites and authentication methods, as based on either signatures or Message Authentication Codes (MACs) calculated from a shared Diffie-Hellman secret. EDHOC has undergone formal verification of its security properties, which was taken as input in the protocol design [22]].

The execution of EDHOC takes place between two peers: one has the role of Initiator, the other the role of Responder (see Figure 2). Based on the exchanged ephemeral Diffie-Hellman keys G^X and G^Y, the two peers derive the shared secret G^{XY}. By additionally exchanging authentication credentials (ID_CRED_I and ID_CRED_R) and MACs

(MAC_2 and MAC_3), EDHOC adds mutual authentication to the basic Diffie-Hellman exchange. By encrypting the exchanged identities with temporary keys, EDHOC protects the Responder identity from passive attacks, and the Initiator identity from active attacks.

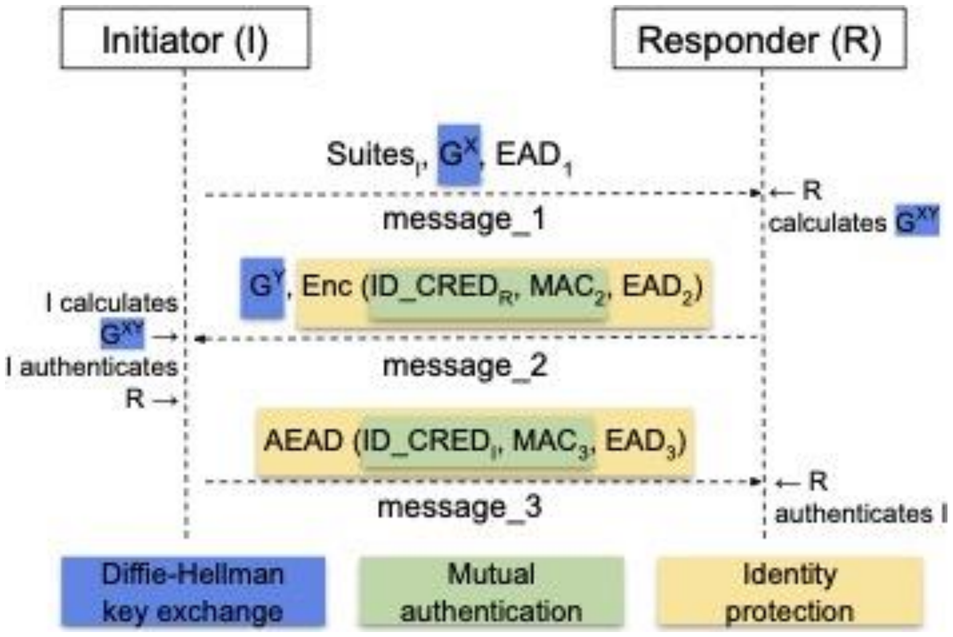


Figure 2. Selected fields of an EDHOC exchange in its static Diffie-Hellman authentication mode. The optional message_4 is omitted. Enc denotes binary additive stream cipher encryption. AEAD stands for authenticated encryption with additional data.

The authorization data fields (EAD1, EAD2 and EAD3) allow external security applications to be integrated in EDHOC without increasing the number of round trips, and enabling additional use of parameters transported in EDHOC. One example is to involve a trusted third party performing online authorization of the interaction between message_1 and message_2, through a voucher requested in EAD_1 and retrieved in EAD_2. This enables the Initiator to simultaneously authenticate and authorize the Responder after the second message. As another example, by including a certificate enrolment request in EAD_3, the Responder is able to authenticate and authorize the Initiator after the third message, and to involve a Certification Authority to request the issue of a C509 operational certificate for the Initiator to use in this particular deployment. The C509 certificate may be returned to the Initiator in message_4 (in the message field EAD_4), or alternatively only a reference is sent back, and the certificate is cached in a repository for non-constrained devices to access without loading the constrained links.

Both examples above are applicable in the industrial scenario to optimize the secure on-boarding of new units in the factory, by performing mutual authentication and authorization, and issuing dedicated device certificates from a factory local Certification Authority in two round trips with minimal overhead.

Design Considerations: One of the main protocol design requirements was low implementation complexity and consequently low code footprint. Meeting this requirement while ensuring that the protocol is usable in a wide range of use cases and deployment scenarios is not trivial. For example, one of the initial goals of the protocol was to support authentication based on symmetric keys, apart from asymmetric authentication credentials. To meet the requirement of low complexity, a design decision was made to rule out authentication based on symmetric keys and focus exclusively on asymmetric credentials.

Another example was the selection of the mandatory to implement cipher suites, which is important from an interoperability point of view, as the common denominator among different applications. An EDHOC cipher suite consists of several algorithms, for authenticated encryption, hashing, key exchange curve and signing. There was no controversy on mandating cipher suites based on Advanced Encryption Standard (AES), as it is a symmetric primitive that is widely implemented in hardware even for constrained devices. Similarly, the SHA-256 algorithm was considered the go-to choice for hashing, due to the many available hardware implementations for constrained devices. The community was split in its preferred choice to mandate an elliptic curve and the corresponding signature algorithm: the question was whether to support the NIST P-256 curve and Elliptic Curve Digital Signature Algorithm (ECDSA), or instead Curve25519 and the Edwards-curve Digital Signature Algorithm (EdDSA). The fact that the EdDSA algorithm is specified with a different hash algorithm than SHA-256 meant that a device implementing EDHOC would have to support two different hash algorithms. This was deemed unacceptable from the code footprint point of view. A decision was made to mandate the NIST P-256 curve and ECDSA, also due to the good support available in constrained hardware.

OSCORE

The protocol Object Security for Constrained RESTful Environments (OSCORE) [7] allows for protecting CoAP messages at the application layer, while enabling those to be forwarded by a proxy (the gateway in our illustrative scenario). OSCORE ensures confidentiality, replay protection, authentication, integrity and ordering of protected messages. In particular, it provides end-to-end security from message producer to message consumer, also in the presence of intermediaries such as proxies.

OSCORE is especially suited for resource-constrained environments, and its design prioritizes small message size and small code footprint, in order to ensure operation in constrained networks. To this end, it relies on CBOR for data encoding and on COSE for security-related services such as encryption. In addition to CoAP, the OSCORE protocol is applicable also to HTTP messages and supports proxy operations for translating between CoAP and HTTP. The security of OSCORE is based on authenticated symmetric encryption.

To communicate using OSCORE, two peers need to first establish an OSCORE Security Context with one another. This can be pre-provisioned to the two peers, or derived by using EDHOC, in which case the two peers start from asymmetric key material to establish a shared secret for keying OSCORE.

The OSCORE Security Context comprises a number of parameters needed for communication with OSCORE. These include: OSCORE identifiers for the two peers, as conceptually distinct from their EDHOC identifiers; settings on the encryption algorithm to use; a counter keeping track of message sequence numbers; a replay window; and cryptographic key information. OSCORE relies on a Master Secret and a Master Salt to generate individual symmetric keys for message protection and unprotection through a key derivation function. Messages in different directions are processed with different keys: a Sender Key for outgoing messages, and a Recipient Key for incoming messages.

OSCORE takes a CoAP message as input and produces a protected CoAP message as output (see Figure 3). In an OSCORE-protected CoAP message, the CoAP method, the payload and most of the CoAP options are encrypted.

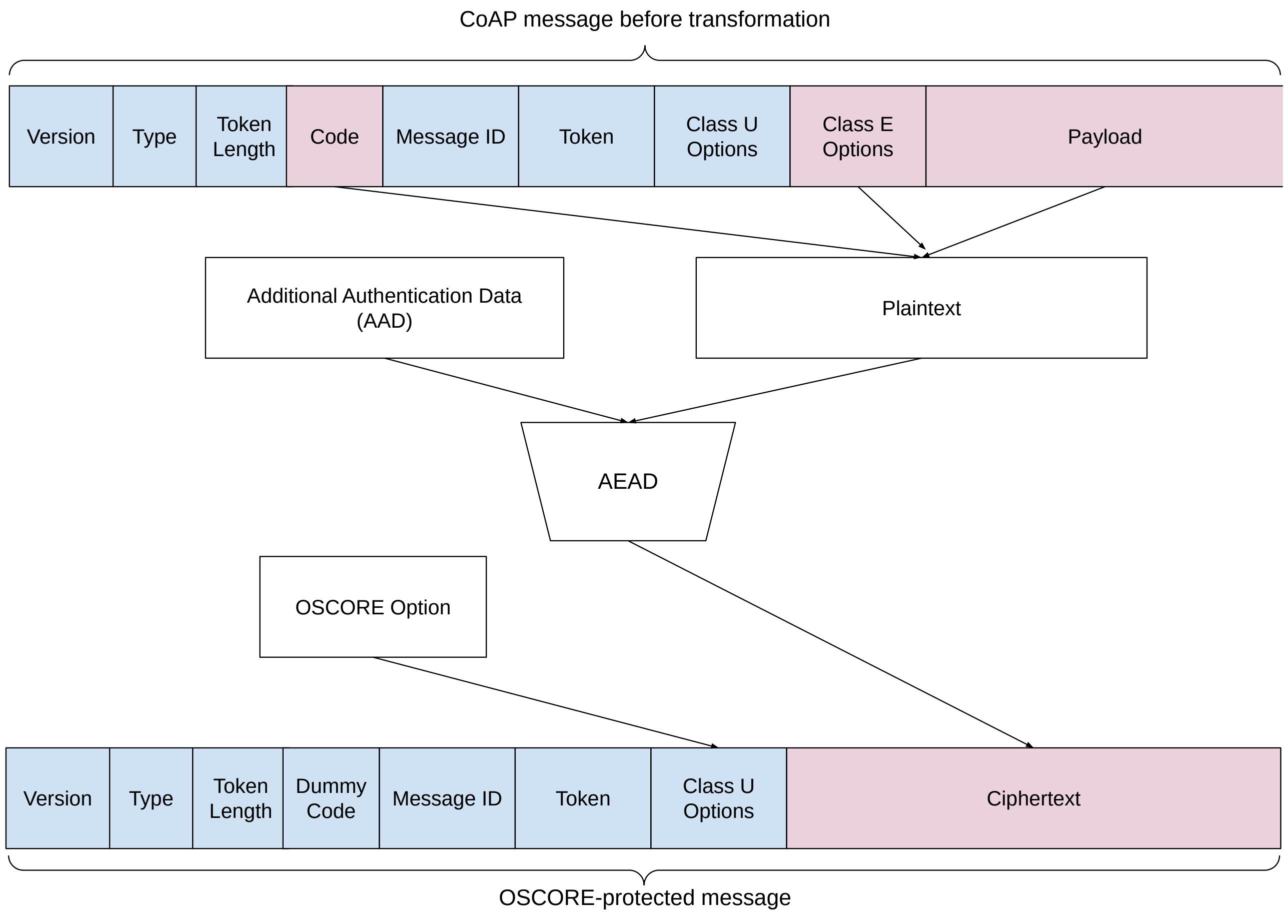


Figure 3. The OSCORE transformation and protection of CoAP messages. AEAD stands for "Authenticated Encryption with Additional Data".

When using EDHOC and OSCORE together, their operation can be optimized (see Figure 4). In case the EDHOC Initiator has data to send immediately after EDHOC has completed, it is possible to use the optimized flow and piggyback EDHOC message_3 with the first OSCORE request. This optimization, defined in an ongoing work [23], reduces the communication overhead by one round trip.

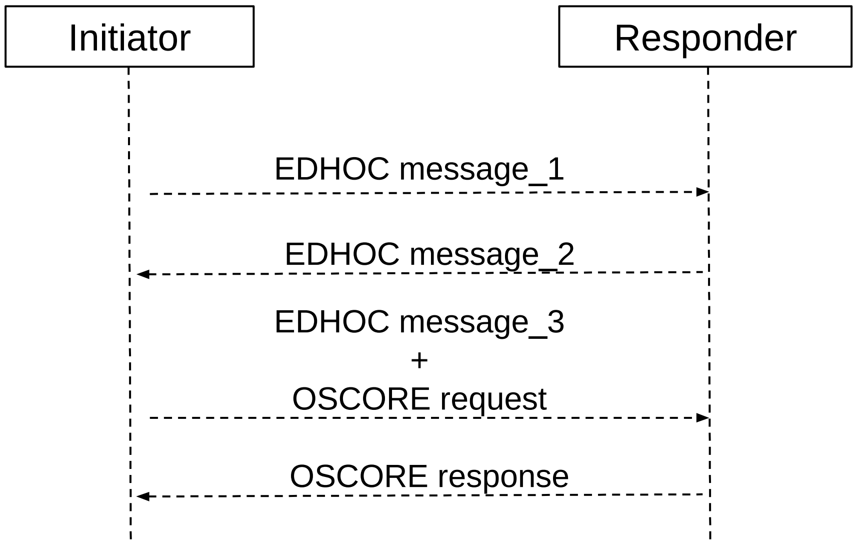


Figure 4. The piggybacking of an OSCORE request with EDHOC message_3 to reduce communication overhead is specified in an ongoing work [23] of the IETF CoRE Working Group.

Design Considerations. The design of OSCORE was led by the motivation to enable end-to-end security in the presence of proxies. This means that the OSCORE security association between two peers does not terminate at a proxy, like in the case of DTLS, but is instead truly end-to-end. A consequence of this design decision is that the proxy has to be able to read the elements of the CoAP message necessary to be able to forward it to the intended recipient. As we show in Figure 3, an OSCORE-protected message is a valid CoAP message including the OSCORE option, which a proxy can seamlessly forward. The OSCORE-protected message, however, hides as much information as possible from the proxy: it encrypts the CoAP code and leaves in the clear only the options necessary for the proxy to see, e.g., those indicating the host part of the resource identifier at the server.

In its initial design, OSCORE was not based on CBOR and COSE, but defined its own encoding and cryptographic processing. While this initial encoding was very efficient, it required custom processing logic for an OSCORE implementation. Later on in the standardization process, as the CBOR and COSE specifications became stable, it was deemed beneficial to rely on them for reducing the overall code footprint of the security stack for constrained devices.

Specification of Lightweight Authorization using EDHOC

The following work is published as an official Internet-Draft specification within the IETF as "Lighweight Authorization using Ephemeral Diffie-Hellman over COSE. Göran Selander, John Mattsson, Mališa Vučinić, Geovane Fedrecheski, Michael Richardson. IETF Internet Draft draft-ietf-lake-authz-01."

Summary

This section describes a procedure for authorizing enrollment of new devices using the lightweight authenticated key exchange protocol Ephemeral Diffie-Hellman Over COSE (EDHOC). The procedure is applicable to zero-touch onboarding of new devices to a constrained network leveraging trust anchors installed at manufacture time.

Introduction

For constrained IoT deployments the overhead and processing contributed by security protocols may be significant, which motivates the specification of lightweight protocols that are optimizing, in particular, message overhead. [8]

This section describes a procedure for augmenting the lightweight authenticated Diffie-Hellman key exchange EDHOC [8] with third party-assisted authorization.

The procedure involves a device, a domain authenticator, and an enrollment server.

The device and domain authenticator perform mutual authentication and authorization, assisted by the enrollment server that provides relevant authorization information to the device (a "voucher") and to the authenticator. The high-level model is similar to BRSKI [18].

In this document, we consider the target interaction for which authorization is needed to be "enrollment", for example joining a network for the first time (e.g., [11]), but it can be applied to authorize other target interactions.

The enrollment server may represent the manufacturer of the device, or some other party with information about the device from which a trust anchor has been pre-provisioned into the device.

The (domain) authenticator may represent the service provider or some other party controlling access to the network in which the device is enrolling.

The protocol assumes that authentication between device and authenticator is performed with EDHOC [8], and defines the integration of a lightweight authorization procedure using the External Authorization Data (EAD) fields defined in EDHOC.

The protocol enables a low message count by performing authorization and enrollment in parallel with authentication, instead of in sequence, which is common for network access. It further reuses protocol elements from EDHOC, leading to reduced message sizes on constrained links. This protocol is applicable to a wide variety of settings, and can be mapped to different authorization architectures.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC2119, RFC8174 when, and only when, they appear in all capitals, as shown here.

Protocol Outline

The goal of this protocol is to enable a (potentially constrained) device (U) to enroll into a domain over a constrained link. The device authenticates and enforces authorization of the (non-constrained) domain authenticator (V) with the help of a voucher conveying authorization information. The voucher has a similar role as in RFC8366 but should be considerably more compact. The domain authenticator, in turn, authenticates the device and authorizes its enrollment into the domain.

The procedure is assisted by a (non-constrained) enrollment server (W) located in a non-constrained network behind the domain authenticator, e.g., on the Internet, providing information to the device (conveyed in the voucher) and to the domain authenticator as part of the protocol. The objective of this document is to specify such a protocol that is lightweight over the constrained link, by reusing elements of EDHOC [8] and by shifting message overhead to the non-constrained side of the network.

See illustration in Figure 5. Note the cardinality of the involved parties. It is expected that the domain authenticator needs to handle a large unspecified number of devices, but for a given device type or manufacturer it is expected that one or a few nodes host enrollment servers.

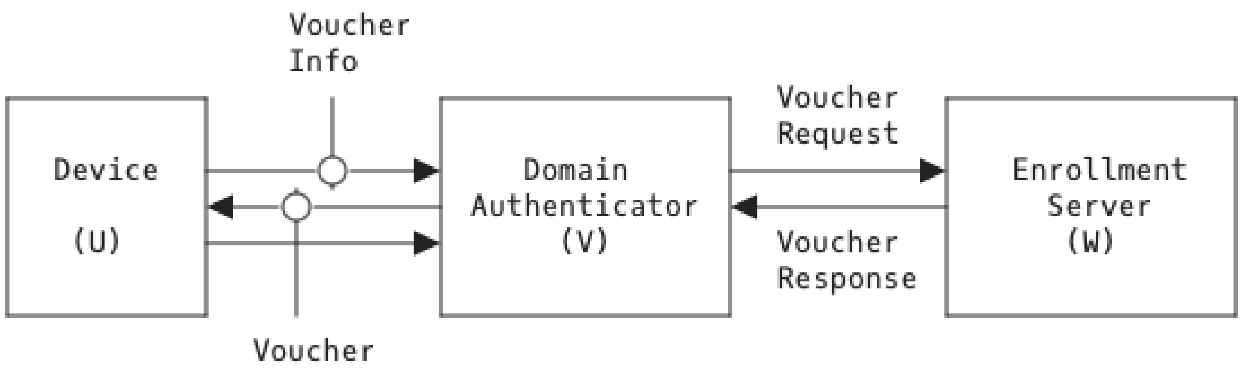


Figure 5. Overview of the message flow. EDHOC is used on the constrained link between U and V. Voucher Info and Voucher are sent in EDHOC External Authorization Data (EAD). The link between V and W is not constrained.

Assumptions

The protocol is based on the following pre-existing relations between the device (U), the domain authenticator (V) and the enrollment server (W), see Figure 6.

- U and W have an explicit relation: U is configured with a public key of W.
- V and W have an implicit relation, e.g., based on web PKI with trusted CA certificates.

- U and V need not have any previous relation. This protocol establishes a relation between U and V.

Each of the three parties can gain protected communication with the other two during the protocol. V may be able to access credentials over non-constrained networks, but U may be limited to constrained networks. Implementations wishing to leverage the zero-touch capabilities of this protocol are expected to support transmission of credentials from V to U by value during the EDHOC exchange, which will impact the message size depending on the type of credential used.

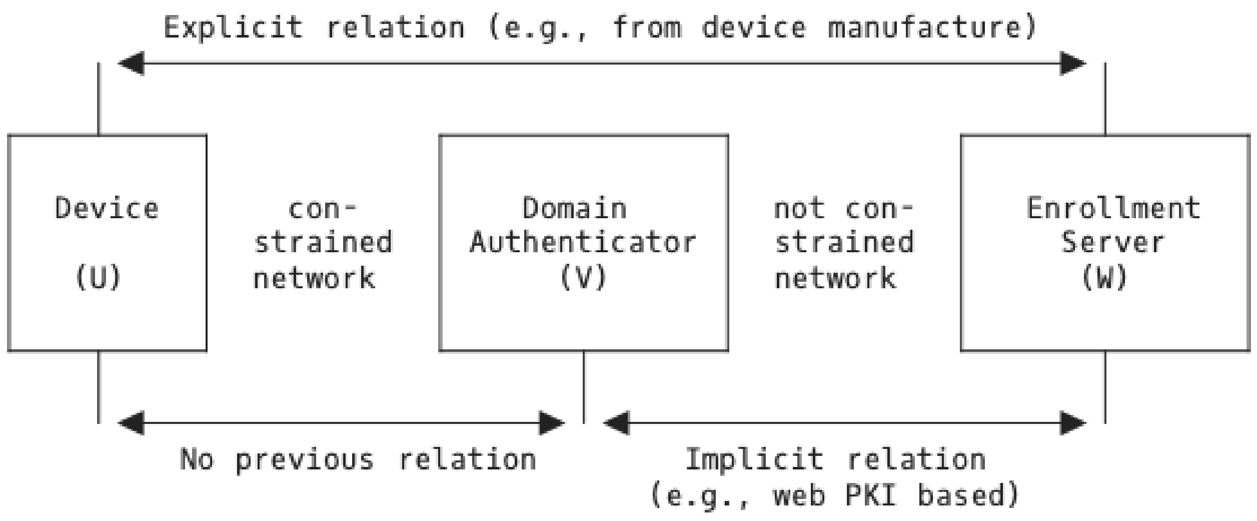


Figure 6. Overview of pre-existing relations.

Device (U)

To authenticate to V, the device (U) runs EDHOC in the role of Initiator with authentication credential CRED_U, for example, an X.509 certificate or a CBOR Web Token (CWT). CRED_U may, for example, be carried by value in ID_CRED_I of EDHOC message_3 or be provisioned to V over a non-constrained network, leveraging a credential identifier in ID_CRED_I (see bottom of Figure 7).

U also needs to identify itself to W. The device identifier used for this is ID_U. The purpose of ID_U is for W to be able to determine if the device with this identifier is authorized to enroll with V. ID_U may be a reference to CRED_U, like ID_CRED_I in

EDHOC (see Section 3.5.2 of [8]), or a device identifier from a different name space, such as EUI-64 identifiers.

U is also provisioned with information about W:

- A static public DH key of W (G_W) used to establish secure communication with the enrollment server.
- Location information about the enrollment server (LOC_W) that can be used by V to reach W. This is typically a URI but may alternatively be only the domain name.

Domain Authenticator (V)

To authenticate to U, the domain authenticator (V) runs EDHOC in the role of Responder with an authentication credential CRED_V, which contains a public key of V. This proves to U the possession of the private key corresponding to the public key of CRED_V. CRED_V typically needs to be transported to U in EDHOC (using ID_CRED_R = CRED_V, see [Section 3.5.2 of [8]) since there is no previous relation between U and V.

V and W need to establish a secure (confidentiality and integrity protected) connection for the Voucher Request/Response protocol. Furthermore, W needs to access the same credential CRED_V that V uses with U, and V needs to prove to W the possession of the private key corresponding to the public key of CRED_V. It is RECOMMENDED that V authenticates to W using the same credential CRED_V as with U.

- V and W may protect the Voucher Request/Response protocol using TLS 1.3 with client authentication if CRED_V is an X.509 certificate of a signature public key. However, note that CRED_V may not be a valid credential to use with TLS 1.3, e.g., when U and V run EDHOC with method 1 or 3, where the public key of CRED_V is a static Diffie-Hellman key.
- V may run EDHOC in the role of initiator with W, using ID_CRED_I = CRED_V. In this case the secure connection between V and W may be based on OSCORE [7].

Note that one solution for establishing secure connection and proof-of-possession is to run TLS 1.3 and EDHOC between V and W during the setup procedure. For example, W may authenticate to V using TLS 1.3 with server certificates signed by a CA trusted by V, and then V may run EDHOC using CRED_V over the secure TLS connection to W, see Figure 7. In this case OSCORE is not needed since the purpose of EDHOC is only to verify proof-of-possession. Note also that the secure connection between V and W may be long lived and reused for multiple voucher requests/responses. Other details of proof-of-possession related to CRED_V and transport of CRED_V are out of scope.

Enrollment Server (W)

The enrollment server (W) is assumed to have the private DH key corresponding to G_W, which is used to establish secure communication with the device. W provides to U the authorization decision for enrollment with V in the form of a voucher.

To calculate the voucher, W needs access to message_1 and CRED_V as used in the EDHOC session between U and V.

- W MUST verify that CRED_V is bound to the secure connection between W and V
- W MUST verify that V is in possession of the private key corresponding to the public key of CRED_V

W needs to be available during the execution of the protocol between U and V.

The Protocol

Overview

The protocol consists of three security sessions going on in parallel:

1. The EDHOC session between device (U) and (domain) authenticator (V)
2. Voucher Request/Response between authenticator (V) and enrollment server (W)

3. An exchange of voucher-related information, including the voucher itself, between device (U) and enrollment server (W), mediated by the authenticator (V).

Figure 7 provides an overview of the message flow detailed in this section. An outline of EDHOC is given in Section 2 of [8].

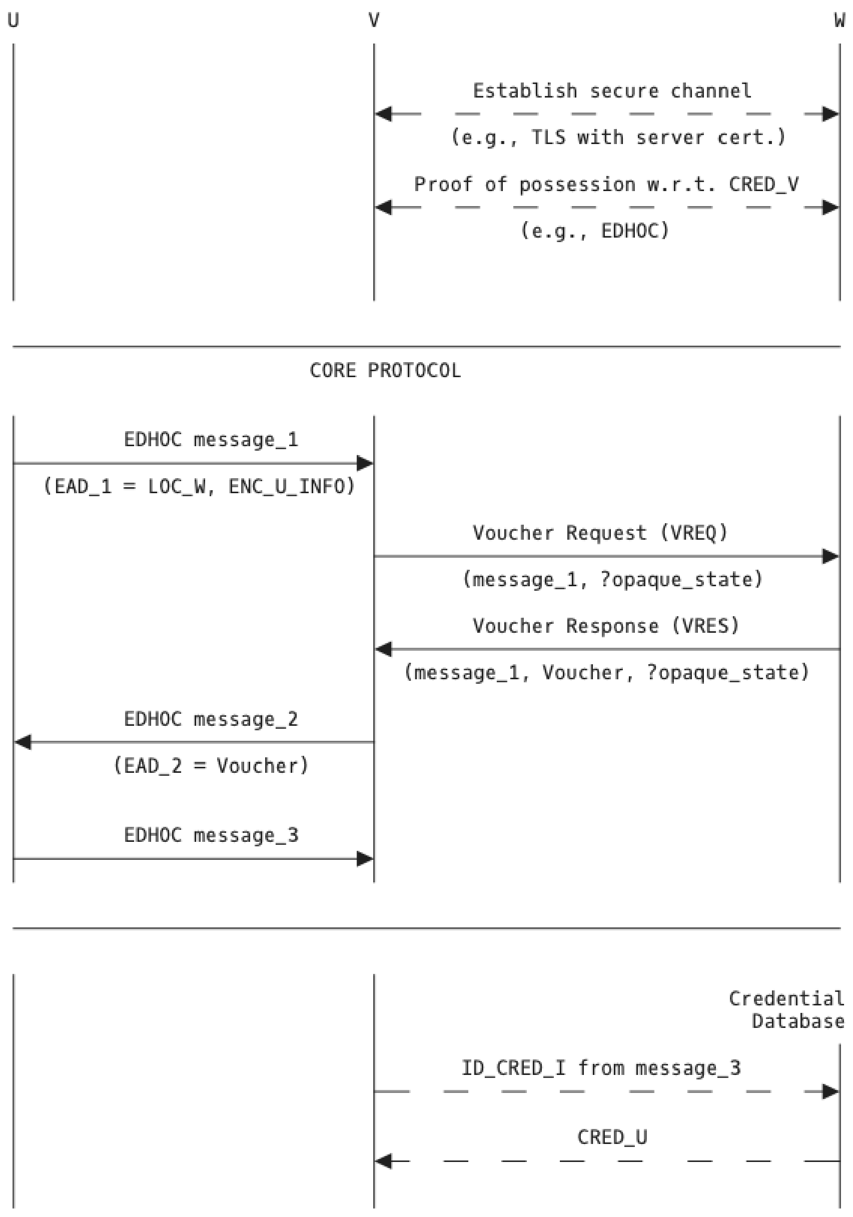


Figure 7. Overview of the protocol: W-assisted authorization of U and V to each other: EDHOC between U and V, and Voucher Request/Response between V and W.

Reuse of EDHOC

The protocol illustrated in Figure 7 reuses several components of EDHOC:

- G_X, the ephemeral public Diffie-Hellman key of U, is also used in the protocol between U and W.
- SUITES_I includes the cipher suite for EDHOC selected by U, and also defines the algorithms used between U and W (see Section 3.6 of [8]:
    - EDHOC AEAD algorithm: used to encrypt ID_U and to generate voucher
    - EDHOC hash algorithm: used for key derivation
    - EDHOC key exchange algorithm: used to calculate the shared secret between U and W
- EAD_1, EAD_2 are the External Authorization Data message fields of message_1 and message_2, respectively, see Section 3.8 of [8]. This document specifies the EAD items with ead_label = TBD1.
- ID_CRED_I and ID_CRED_R are used to identify the authentication credentials CRED_U and CRED_V, respectively. As shown at the bottom of Figure 7, V may use W to obtain CRED_U. CRED_V is transported in ID_CRED_R in message_2.

The protocol also reuses the EDHOC_Extract and EDHOC_Expand key derivation from EDHOC (see Section 4 of [8]).

- The intermediate pseudo-random key PRK is derived using EDHOC_Extract():
    - PRK = EDHOC_Extract(salt, IKM)
        - where salt = 0x (the zero-length byte string)
        - IKM is computed as an ECDH cofactor Diffie-Hellman shared secret from the public key of W, G_W, and the private key corresponding to G_X (or v.v.), see Section 5.7.1.2 of NIST-800-56A.

The output keying material OKM is derived from PRK using EDHOC_Expand(), which is defined in terms of the EDHOC hash algorithm of the selected cipher suite, see Section 4.1.2 of [8]:

- OKM = EDHOC_Expand(PRK, info, length)

where

info = (

  info_label : int,

  context : bstr,

  length : uint,

)

Stateless Operation of V

V may act statelessly with respect to U: the state of the EDHOC session started by U may be dropped at V until authorization from W is received. Once V has received EDHOC message_1 from U and extracted LOC_W from EAD_1, message_1 is forwarded unmodified to W in the form of a Voucher Request. V encapsulates the internal state that it needs to later respond to U, and sends that to W together with EDHOC message_1. This state typically contains addressing information of U (e.g., U's IP address and port number), together with any other implementation-specific parameter needed by V to respond to U. At this point, V can drop the EDHOC session that was initiated by U.

The encapsulated state MUST be protected using a uniformly-distributed (pseudo-)random key, known only to itself and specific for the current EDHOC session to prevent replay attacks of old encapsulated state. How V serializes and encrypts its internal state is out of scope.

W sends to V the voucher together with the echoed message_1, as received from U, and V's internal state. This allows V to act as a simple message relay until it has obtained the authorization from W to enroll U. The reception of a successful Voucher Response at V from W implies the authorization for V to enroll U. At this point, V can initialize a new EDHOC session with U, based on the message and the state retrieved from the Voucher Response from W.

Device <-> Enrollment Server (U <-> W)

The protocol between U and W is carried between U and V in message_1 and message_2, and between V and W in the Voucher Request/Response. The data is protected between the endpoints using secret keys derived from a Diffie-Hellman shared secret as further detailed in this section.

Voucher Info

The external authorization data EAD_1 contains an EAD item with ead_label = TBD1 and ead_value = Voucher_Info, which is a CBOR byte string:

```
Voucher_Info = bstr .cbor Voucher_Info_Seq

Voucher_Info_Seq = (

   LOC_W:      tstr,

   ENC_U_INFO:     bstr

)
```

where

LOC_W is a text string used by V to locate W, e.g., a URI or a domain name.

ENC_U_INFO is a byte string containing an encrypted identifier of U and, optionally, opaque application data prepared by U. It is calculated as follows:

ENC_U_INFO is encrypted using the EDHOC AEAD algorithm of the selected cipher suite specified in SUITE_I of EDHOC message_1. It consists of 'ciphertext' of COSE_Encrypt0 (Section 5.2 of [RFC9052]) computed from the following:

The encryption key K_1 and nonce IV_1 are derived as specified below.

'protected' is a byte string of size 0

'plaintext' and 'external_aad' as below:

```
plaintext = (
    ID_U:           bstr,
    ?OPAQUE_INFO:   bstr,
)

external_aad = (
    SS:             int,
)
```

where

ID_U is an identifier of the device.

OPAQUE_INFO is an opaque field provided by the application. If present, it will contain application data that U may want to convey to W, e.g., enrollment hints. Note that OPAQUE_INFO is opaque when viewed as an information element in EDHOC. It is opaque to V, while the application in U and W can read its contents. The same applies to other references of OPAQUE_INFO throughout this document.

SS is the selected cipher suite in SUITES_I of EDHOC message_1.

The external_aad is wrapped in an enc_structure as defined in Section 5.3 of RFC9052.

The derivation of K_1 = EDHOC_Expand(PRK, info, length) uses the following input to the info struct:

info_label = 0

context = h'' (the empty CBOR string)

length is length of the key of the EDHOC AEAD algorithm in bytes (which is the length of K_1)

The derivation of IV_1 = EDHOC_Expand(PRK, info, length) uses the following input to the info struct (see OKM in Section 4.2):

info_label = 1

context = h'' (the empty CBOR string)

length is length of the nonce of the EDHOC AEAD algorithm in bytes (which is the length of IV_1)

Voucher

The voucher is an assertion to U that W has authorized V. It is encrypted using the EDHOC AEAD algorithm of the selected cipher suite specified in SUITE_I of EDHOC message_1. It consists of the 'ciphertext' field of a COSE_Encrypt0 object:

Voucher = COSE_Encrypt0.ciphertext

Its corresponding plaintext value consists of an opaque field that can be used by W to convey information to U, such as a voucher scope. The authentication tag present in the ciphertext is also bound to message_1 and the credential of V as described below.

The encryption key K_2 and nonce IV_2 are derived as specified below.

'protected' is a byte string of size 0

'plaintext' and 'external_aad' as below:

```
plaintext = (
   ?OPAQUE_INFO: bstr
)
external_aad = (
   H(message_1):  bstr,
   CRED_V:       bstr,
)
```

where

OPAQUE_INFO is an opaque field provided by the application.

H(message_1) is the hash of EDHOC message_1, calculated from the associated voucher request. The hash is computed by using the EDHOC hash algorithm of the selected cipher suite specified in SUITE_I of EDHOC message_1.

CRED_V is the CWT Claims Set containing the public authentication key of V.

The derivation of K_2 = EDHOC_Expand(PRK, info, length) uses the following input to the info struct:

info_label = 2

context = h'' (the empty CBOR string)

length is length of the key of the EDHOC AEAD algorithm in bytes

The derivation of IV_2 = EDHOC_Expand(PRK, info, length) uses the following input to the info struct:

info_label = 3

context = h'' (the empty CBOR string)

length is length of the nonce of the EDHOC AEAD algorithm in bytes.

Device <-> Authenticator (U <-> V)

This section describes the processing in U and V, which includes the EDHOC protocol, see Figure 7. Normal EDHOC processing is omitted here.

Message 1

Processing in U

U composes EDHOC message_1 using authentication method, identifiers, etc. according to an agreed application profile, see Section 3.9 of [8]. The selected cipher suite, in this document denoted SS, applies also to the interaction with W, in particular, with respect to the Diffie Hellman key agreement algorithm used between U and W. As part of the normal EDHOC processing, U generates the ephemeral public key G_X that is reused in the interaction with W.

The device sends EDHOC message_1 with EAD item (-TBD1, Voucher_Info) included in EAD_1,The negative sign indicates that the EAD item is critical, see Section 3.8 of [8].

Processing in V

V receives EDHOC message_1 from U and processes it as specified in Section 5.2.3 of [8], with the additional step of processing the EAD item in EAD_1. Since the EAD item is critical, if V does not recognize it or it contains information that V cannot process, then V MUST abort the EDHOC session, see Section 3.8 of [8]. Otherwise, the ead_label = TBD1 triggers the voucher request to W. The exchange between V and W needs to be completed successfully for the EDHOC session to be continued.

Message 2

Processing in V

V receives the voucher response from W. V sends EDHOC message_2 to U with the critical EAD item (-TBD1, Voucher) included in EAD_2, i.e., ead_label = TBD1 and ead_value = Voucher.

CRED_V is a CWT Claims Set [RFC8392] containing the public authentication key of V encoded as a COSE_Key in the 'cnf' claim, see Section 3.5.2 of [8]. ID_CRED_R contains the CWT Claims Set with 'kccs' as COSE header_map, see Section 10.6 of [8].

Processing in U

U receives EDHOC message_2 from V and processes it as specified in Section 5.3.3 of [8], with the additional step of processing the EAD item in EAD_2.

If U does not recognize the EAD item or the EAD item contains information that U cannot process, then U MUST abort the EDHOC session, see Section 3.8 of [8]. Otherwise, U MUST verify the Voucher using H(message_1), CRED_V, and the keys derived. If the verification fails, then U MUST abort the EDHOC session.

Message 3

Processing in U

If all verifications passed, then U sends EDHOC message_3.

EDHOC message_3 may be combined with an OSCORE-protected application request.

Processing in V

V performs the normal EDHOC verifications of message_3. V may retrieve CRED_U from a Credential Database, after having learnt ID_CRED_I from U.

Authenticator <-> Enrollment Server (V <-> W)

It is assumed that V and W have set up a secure connection, W has accessed the authentication credential CRED_V to be used in the EDHOC session between V and U, and that W has verified that V is in possession of the private key corresponding to CRED_V. V and W run the Voucher Request/Response protocol over the secure connection.

Voucher Request

Processing in V

V sends the voucher request to W. The Voucher Request SHALL be a CBOR array as defined below:

```
Voucher_Request = [
   message_1:     bstr,
   ? opaque_state: bstr
]
```

where

message_1 is a CBOR byte string whose value is the byte serialization of EDHOC message_1 as it was received from U.

opaque_state is OPTIONAL and represents the serialized and encrypted opaque state needed by V to statelessly respond to U after the reception of Voucher_Response.

Processing in W

W receives and parses the voucher request received over the secure connection with V. The voucher request essentially contains EDHOC message_1 as sent by U to V. W SHALL NOT process message_1 as if it was an EDHOC message intended for W.

W extracts from message_1:

SS - the selected cipher suite, which is the (last) integer of SUITES_I.

G_X - the ephemeral public key of U

ENC_U_INFO - the encryption of (1) the device identifier ID_U and (2) the optional OPAQUE_INFO field, contained in the Voucher_Info field of the EAD item with ead_label = TBD1 (with minus sign indicating criticality)

W verifies and decrypts ENC_U_INFO using the relevant algorithms of the selected cipher suite SS (see Section 4.2), and obtains ID_U.

In case OPAQUE_INFO is present, it is made available to the application.

W calculates the hash of message_1 H(message_1), and associates this session identifier to the device identifier ID_U. W uses ID_U to look up the associated authorization policies for U and enforces them, which is out of scope..

Voucher Response

Processing in W

W retrieves CRED_V associated with the secure connection with V, and constructs the Voucher for the device with identifier ID_U.

W generates the voucher response and sends it to V over the secure connection. The Voucher_Response SHALL be a CBOR array as defined below:

```
Voucher_Response = [
   message_1:     bstr,
   Voucher:       bstr,
   ? opaque_state: bstr
]
```

where

message_1 is a CBOR byte string whose value is the byte serialization of EDHOC message_1 as it was received from V. opaque_state is the echoed byte string opaque_state from Voucher_Request, if present.

Processing in V

V receives the voucher response from W over the secure connection. If present, V decrypts and verifies opaque_state as received from W. If that verification fails, then the EDHOC session with U is aborted. If the voucher response is successfully received from W, then V responds to U with EDHOC message_2.

Error Handling

dThis section specifies a new EDHOC error code and how it is used in the proposed protocol.

EDHOC Error “Access Denied”

This section specifies the new EDHOC error "Access denied", see Table 3.

Table 3. EDHOC error code and error information for "Access denied".

ERR_CODE	ERR_INFO Type	Description
TBD3	error_content	Access denied

Error code TBD3 is used to indicate to the receiver that access control has been applied and the sender has aborted the EDHOC session. The ERR_INFO field contains error_content which is a CBOR Sequence consisting of an integer and an optional byte string.

```
error_content = (
  REJECT_TYPE : int,
  ? REJECT_INFO : bstr,
)
```

The purpose of REJECT_INFO is for the sender to provide verifiable and actionable information to the receiver about the error, so that an automated action may be taken to enable access.

Table 4. REJECT_TYPE and REJECT_INFO for "Access denied".

REJECT_TYPE	REJECT_INFO	Description
0	-	No REJECT_INFO
1	bstr	REJECT_INFO from trusted third party

Error handling in W, V, and U

This protocol uses the EDHOC Error "Access denied" in the following way:

W generates error_content and transfers it to V via the secure connection. If REJECT_TYPE is 1, then REJECT_INFO is encrypted from W to U using the EDHOC AEAD algorithm.

V receives error_content, prepares an EDHOC "Access denied" error, and sends it to U.

U receives the error message and extracts the error_content. If REJECT_TYPE is 1, then U decrypts REJECT_INFO, based on which it may retry to gain access.

The encryption of REJECT_INFO follows a procedure analogous to the one defined in Section Voucher, with the following differences:

```
plaintext = (
    OPAQUE_INFO:     bstr,
 )
external_aad = (
    H(message_1):    bstr,
 )
```

where

OPAQUE_INFO is an opaque field that contains actionable information about the error. It may contain, for example, a list of suggested Vs through which U should join instead. H(message_1) is the hash of EDHOC message_1, calculated from the associated voucher request, see Section 4.6.1.

Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments

The following work was published as "Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments". Geovane Fedrecheski, Mališa Vučinić, Thomas Watteyne. IEEE Wireless Communications and Networking Conference (WCNC), Dubai, United Arab Emirates, 21-24 April 2024."

Summary

Authenticated key exchange protocols play a crucial role in the communication security stack of an Internet-of-Things (IoT) device: they authenticate the communicating parties

and establish a shared symmetric secret between them. Following a large debate in the community, the Internet Engineering Task Force (IETF) has recently standardized a new protocol called EDHOC for authenticated key exchange targeting IoT environments. The EDHOC protocol performs a compact Diffie-Hellman key exchange handshake, requiring several times less bytes-over-the-air than the de-facto solution used in the Internet, the (D)TLS protocol. In this paper, we study how this reduction in message size correlates with the usage of other scarce resources in IoT environments: time, energy, and memory. We evaluate EDHOC and DTLS with different authentication configurations over two IoT radio technologies. First, we measure the EDHOC and DTLS handshakes on constrained hardware over an IEEE 802.15.4 radio. We observe that EDHOC achieves x6 to x14 reduction in packet sizes, x1.44 improvement in handshake duration and x2.79 reduction in energy consumed. Next, we simulate time on air on LoRaWAN networks and find that, in the most restrictive configuration (SF=12), DTLS uses at least x7 more time on air than EDHOC. Finally, we measure flash memory and RAM usage, with the EDHOC implementation achieving a x4 reduction in both.

Evaluation

We present the evaluation of the EDHOC and DTLS handshake in three steps. First, we evaluate bytes over the air, handshake duration, and energy consumption on constrained IoT hardware and using an IEEE 802.15.4 radio. Next, we use an airtime model to simulate the duration of the handshake in LoRa networks with varying spreading factors. Finally, we assess memory usage of the implementation.

IEEE 802.15.4

We evaluate EDHOC and DTLS over IEEE 802.15.4, using a network stack composed of IEEE 802.15.4, 6LoWPAN, IPv6, and UDP. We use IEEE 802.15.4 in its non-beacon mode and the unslotted CSMA-CA multiple access technique. In the case of EDHOC, we also include the CoAP layer and use it for EDHOC transport. We leverage RIOT, an IoT operating system, and its gnrc network implementation.

To evaluate EDHOC, we use the lakers Rust crate, linked as a static library in RIOT. To evaluate DTLS, we use wolfSSL, an embedded-friendly library with support for DTLS version 1.3. We evaluate EDHOC and DTLS libraries running on nRF52840 hardware clocked at 64 MHz. nRF52840 is a 32-bit ARM Cortex-M4 system on chip that integrates an IEEE 802.15.4-compatible radio and a cryptographic accelerator, the ARM Crypto Cell 310 (CC310). We configure the EDHOC library to use the authentication method 3 and cipher suite 2. EDHOC method 3 corresponds to authentication using static Diffie-Hellman keys on both ends. EDHOC cipher suite 2 implies Elliptic-curve Diffie-Hellman key exchange (ECDHE) on curve P-256, SHA-256 hash function, and AES-128-CCM authenticated encryption algorithm. In the absence of static Diffie-Hellman key authentication in DTLS 1.3, we configure the DTLS 1.3 library to use ECDHE authenticated with elliptic-curve digital signature algorithm (ECDSA). We rely on the same elliptic curve (P-256) and authenticated encryption algorithm (AES-128-CCM) as in the case of EDHOC. In both cases, all cryptographic operations are hardware-accelerated using CC310 cryptographic backend.

We evaluate a total of five configurations: EDHOC with RPKs and mutual authentication, DTLS with RPKs with/without mutual authentication, and DTLS with certificates with/without mutual authentication. Note that the mutual authentication is mandatory in EDHOC but optional in DTLS.

Bytes over the air: We assess the number of bytes over the air when considering the full network stack, which includes headers from lower layers and fragmentation. To do so, we run the handshake between two devices and sniff the radio traffic with a third device, and analyze the capture.

We summarize the results in Figure 8. We can see that EDHOC sends ×6.09 and ×14.35 less bytes over the air when compared to DTLS with raw public keys without mutual authentication and to DTLS with certificates and mutual authentication, respectively. One aspect to note is that in the DTLS case, the accumulated headers of the MAC layer already top the accumulated bytes of the full EDHOC handshake. The reason is that although the IEEE 802.15.4 headers are only 21 bytes, its Maximum Transmission Unit (MTU) is 127 bytes, causing DTLS to rely heavily on fragmentation, as shown in Table 5.

Table 5. Messages and fragments per handshake, and the number of bytes over the air, for each layer.

Protocol	Messages	Fragments	IEEE 802.15.4	6LoWPAN	CoAP	Content	Total
EDHOC RPK Mutual	4	4	84	36	66	103	289
DTLS RPK	10	18	378	154	0	1228	1760
DTLS RPK Mutual	13	23	483	199	0	1558	2240
DTLS Cert	10	26	546	194	0	1961	2701
DTLS Cert Mutual	13	39	819	279	0	3049	4147

Thus, the MAC headers are sent many times over the network, further accentuating the message sizes differences. Table 5 also shows the accumulated bytes over the air per layer of the network stack. Note that 6LoWPAN contains the compressed IPv6 and UDP headers. Handshake duration and energy consumption: Next, we look into the question of how the difference in bytes over the air translates into the usage of other resources: handshake duration and energy consumption. We measure it using a power profiler device (Otii Arc4) connected to the device that initiates the handshake. The power profiler provides power to the device under test (DUT), and measures its current consumption. The profiler also supports reading from a digital pin, allowing us to precisely time each handshake. The DUT firmware was configured to run the handshake 20 times. We collect the results using the power profiler software and calculate the amount of energy and time spent for each handshake.

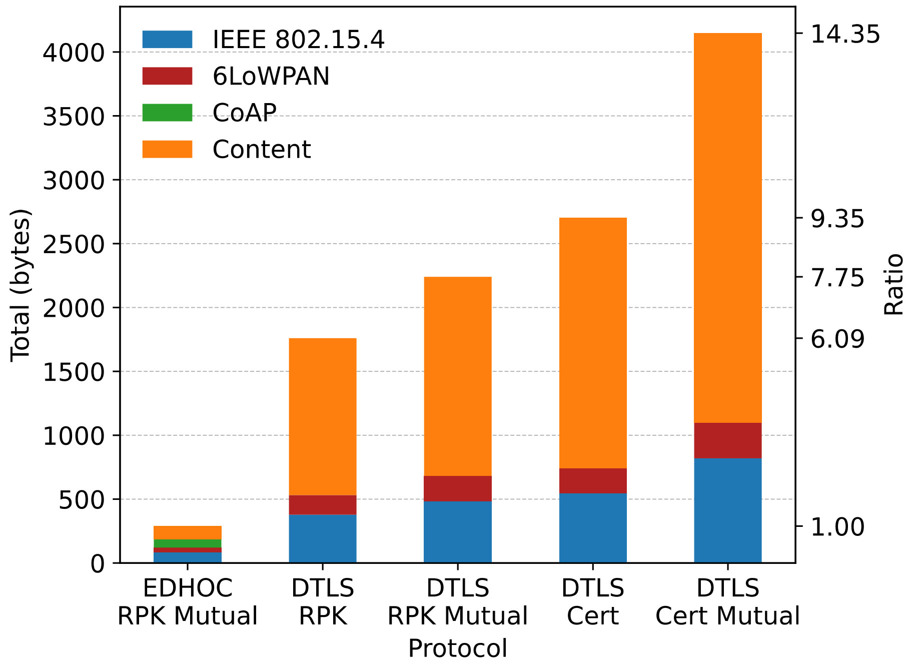
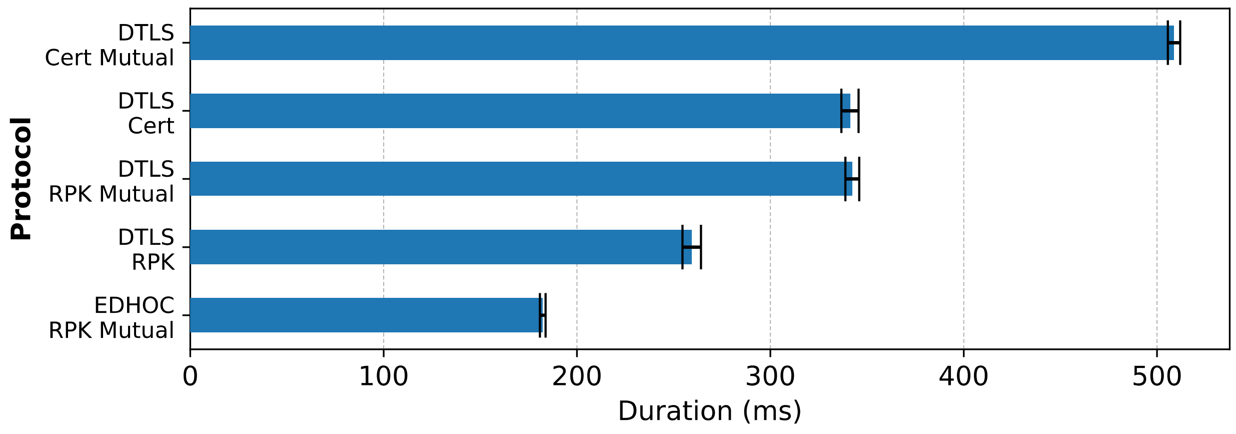


Figure 8. Bytes over the air for EDHOC and DTLS handshakes.

Figure 9 presents the results on handshake duration and energy consumption. We note that the duration and energy are highly correlated. The configuration of DTLS with raw public keys and no mutual authentication uses approximately ×1.44 more time and energy compared to EDHOC. The DTLS configuration with certificates and mutual authentication uses ×2.79 more time and energy compared to EDHOC. Thus, the improvement in packet sizes in EDHOC actually translates to a smaller improvement in duration and energy consumption. Note that the smaller packets are due to the compact serialization used by EDHOC, which imply less radio usage but not necessarily processing time, the latter being heavily affected by cryptographic operations. This is relevant since both protocols feature CPU-intensive computations, most notably the generation of ephemeral asymmetric key pairs and performing a Diffie-Hellman operation.



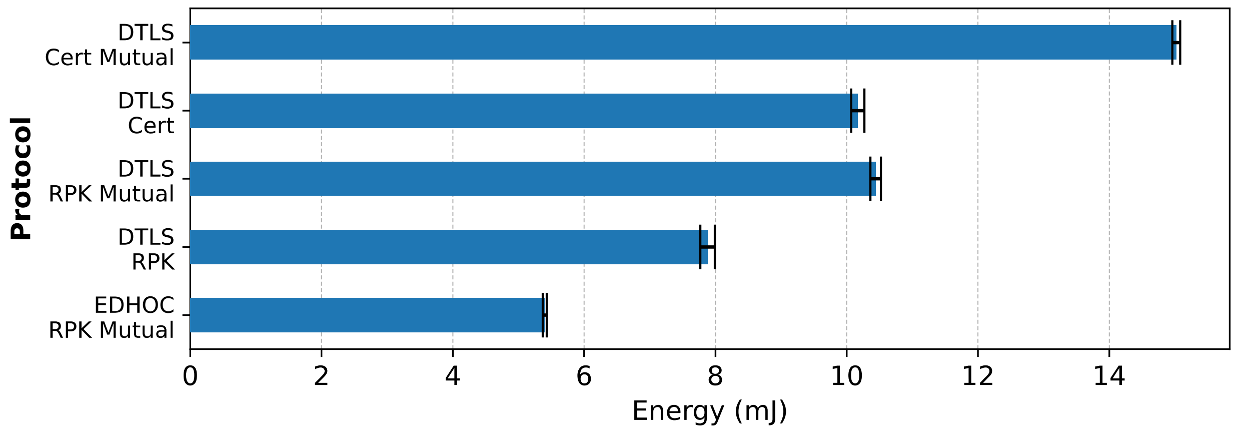


Figure 9. Results of time and energy measurements for a handshake under several configurations.

LoRaWAN

We evaluated the time on air (ToA) for executing EDHOC and DTLS handshakes over LoRaWAN. LoRa is a low-power, long-range technology designed to run on ISM bands. Its corresponding link layer is called LoRaWAN, and it defines the communications protocol and architecture. LoRaWAN is highly configurable, allowing clients to choose a particular spreading factor and bandwidth that better suits their needs for data rate, distance, and energy consumption. To prevent channel overuse, regulators impose severe radio duty cycles, typically in the order of 1%. Although there are efforts to bring IP connectivity to LoRaWAN networks, most implementations transmit application data directly on top of the link layer, due to LoRaWAN's data rate constraints. EDHOC can be used in this scenario, since it was designed to be independent of the underlying transport. Running DTLS on top of a LoRaWAN link, however, becomes impractical, since DTLS assumes an underlying UDP/IP stack. For this reason, similarly to other works [24]], we approached the problem of evaluating DTLS over LoRaWAN by using a simulation based on an airtime model. We collected messages from a real handshake over IEEE 802.15.4, stripped off the link layer headers and, after calculating the resulting message sizes, divided them in lists of fragment sizes according to the MTU of each LoRaWAN spreading factor (SF). We then used an open source tool to estimate the time on air for EDHOC and DTLS over LoRaWAN in several spreading factor configurations.

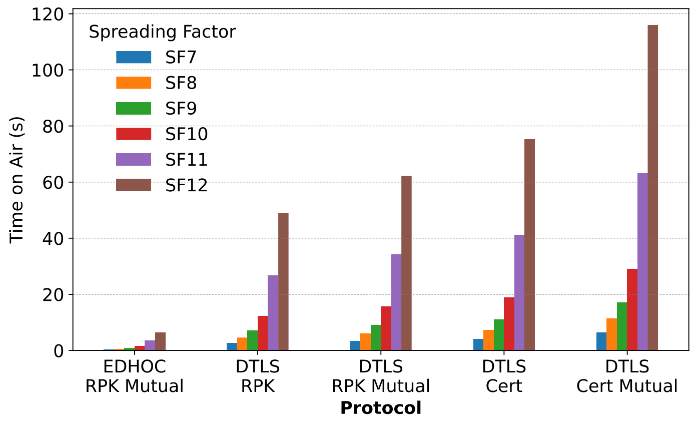
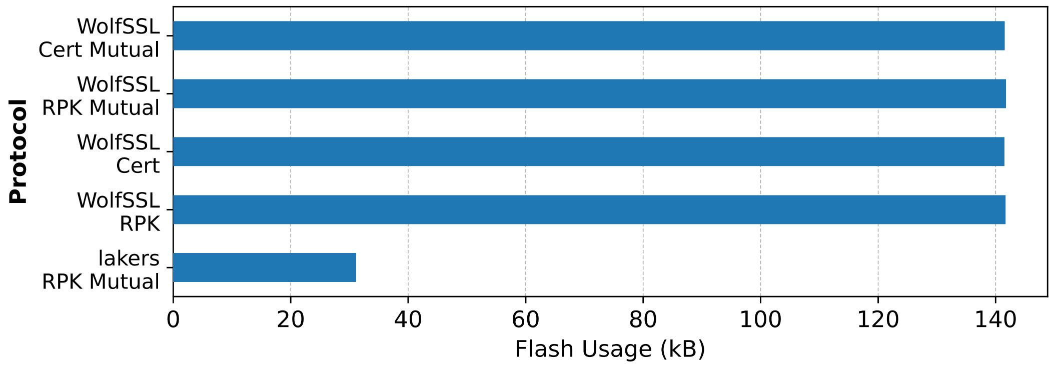


Figure 10. Time on air for EDHOC and DTLS handshakes under different spreading factors (not considering duty cycle).

Figure 10 presents the results. With Spreading Factor 12, the time on air for EDHOC is less than 7 seconds, while for DTLS it can be almost 50 or 80 seconds for RPK and certificates, respectively.

Memory Usage

We measure memory usage in two steps. First, we measure flash memory and static RAM by analyzing the .map files generated during compilation. In this analysis we only include the symbols relevant to the library (either lakers for EDHOC or WolfSSL for DTLS) and its application code, i.e., symbols used by the operating system and crypto backend are discarded. In the second step, we measure peak stack and heap to obtain runtime usage of the RAM. We use the RIOT module ps to measure stack and implement heap-painting to obtain maximum heap usage. Note that peak heap is only measured for WolfSSL, since lakers does not use dynamic memory.



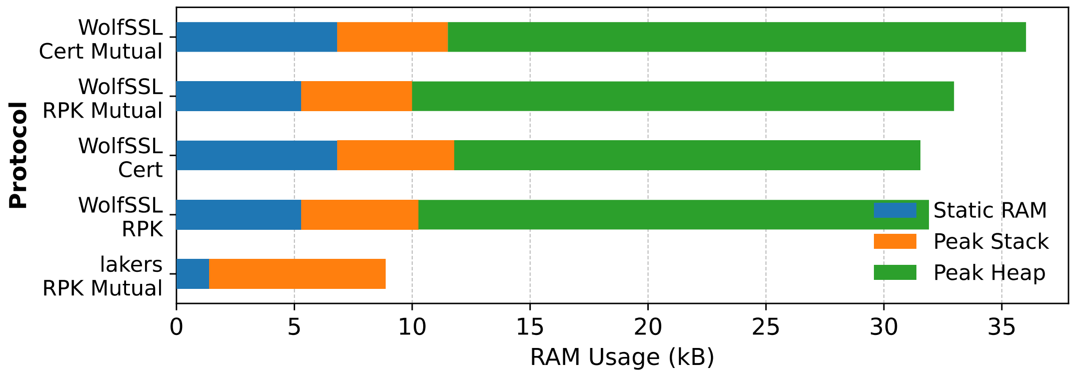


Figure 11. Flash memory and RAM usage for the lakers (implementation of EDHOC) and WolfSSL (implementation of DTLS) implementations.

Flash memory: Figure 11 presents the RAM and flash footprint results. wolfSSL uses virtually the same amount of flash in all configurations, and requires about ×4.5 more flash than lakers. This is in part due to the fact that EDHOC was designed to be simple to parse and decode, which it does by relying on CBOR encoding. In addition, the lakers implementation uses only inline CBOR processing, i.e., it does not depend on a CBOR library. It is also worth noting that the lakers implementation only implements one authentication method and cipher suite, while wolfSSL supports several DTLS configurations. A final nuance is that, while wolfSSL is implemented in C, lakers is written in Rust, which typically leads to binaries twice as large than C [25]]. Volatile memory: Figure 11 also presents the results on RAM footprint. Overall, WolfSSL uses between ×3.6 and ×4 more RAM than lakers. The lakers implementation uses far less static RAM, but almost double of the stack, which is compensated by its zero use of the heap. Among the reasons for this difference, are the increased complexity of parsing DER files versus the more simple CBOR approach, and the fact that DTLS needs a more elaborate state machine due to the amount of messages that are exchanged (see Table 5).

Benchmarking Results of a Zero-touch lifecycle framework

The following work is under submission as "Secure IoT enrollment: lightweight, zero-touch, and under standardization. Geovane Fedrecheski, Mališa Vučinić, Göran Selander, Thomas Watteyne. IEEE Internet of Things Journal, to appear in 2024."

Implementation

We implement and publish EDHOC with Lightweight Authorization (ELA) as an open-source Rust crate, integrated within the lakers EDHOC library[5]. To expand the range of platforms where ELA can be run, we also provide bindings for C and Python. We also implement the enrollment server (W) to enable fully testing the protocol. W is written in Python (leveraging the provided Python bindings) and offers a web-based Graphical User Interface (GUI) for demonstration purposes.

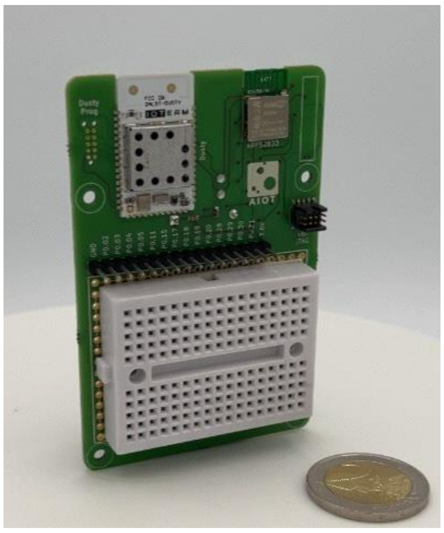
Evaluation

We evaluate ELA running on top of two hardware platforms and the two corresponding networking technologies (see Figure 12 and Figure 13). We consider message size, execution time, energy consumption, and memory usage as the evaluation metrics. We compare ELA against EAP-EDHOC [17], a state-of-the-art protocol under standardization in the IETF. The EAP-EDHOC protocol [17] specifies how to use EDHOC

[5] https://github.com/openwsn-berkeley/lakers

to authenticate peers using EAP [14], a widely used authentication protocol that is applicable to network join.

Since both ELA and EAP-EDHOC authenticate using EDHOC, both protocols achieve similar levels of protection. Similarly to ELA, EAP-EDHOC is independent of the underlying network layer, and can be deployed from constrained to non-constrained networks. Unlike ELA, EAP-EDHOC can be deployed in networks that support multiple EAP methods, thus enjoying greater flexibility. Given that both protocols solve similar problems but with slight differences in flexibility, we compare the associated costs in terms of message sizes and execution time.



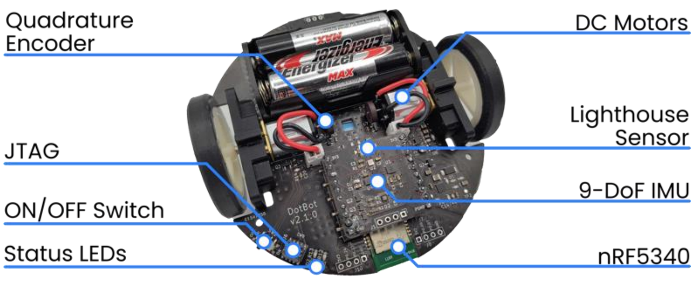


Figure 12. The DotBot swarm robotic platform used to evaluated ELA.

Figure 13. The AIOT Plat platform gathers an nRF52833 Microcontroller (MCU) and a SmartMesh IP radio module used for the evaluation of ELA.

Use Case 1: ELA and EAP-EDHOC over SmartMesh IP

We first compare packet sizes of ELA and EAP-EDHOC. We then execute both protocols on a duty cycled network to understand the average duration of each protocol.

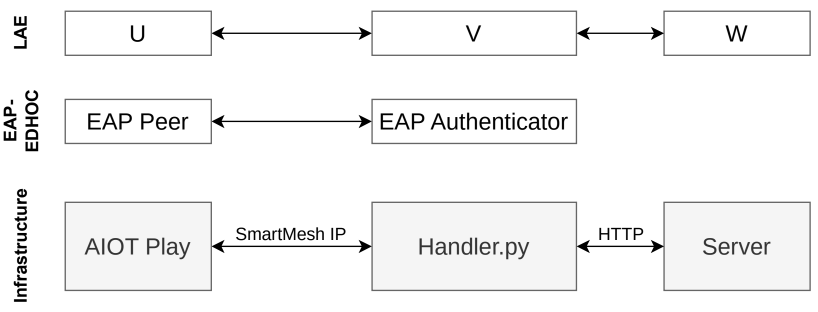


Figure 14. ELA and EAP-EDHOC integrated with the SmartMesh IP infrastructure.

Platform

The AIOT Play platform was developed for educational purposes and is used for teaching in France. We leverage it as an example of a typical IoT device with a constrained MCU and a low-power constrained Internet-of-Things radio technology. The platform integrates an nRF52833 MCU, a SmartMesh IP radio module [26], and a prototyping area. The MCU features a 64-MHz processor with 128 kB of RAM and 512 kB of flash memory. Our code implementing ELA is executed on the nRF52833 MCU. The MCU hands over the serial port each message to be transmitted to the SmartMesh IP radio module. The SmartMesh IP radios form a wireless multi-hop mesh network. The SmartMesh IP radio module implements a networking stack based on Time-synchronized Channel Hopping (TSCH) technology; an industry-vetted solution used in factory monitoring use cases worldwide. In our case, a single hop network is used for the evaluation.

Message Sizes

To compare message footprints, we implemented a packet encoder for EAP-EDHOC and generated a set of packets corresponding to a successful message flow [17]. The EAP-EDHOC headers were built using our encoder, while the EDHOC message payloads (when applicable) were generated by the lakers library and fed into the EAP encoder. Packets corresponding to a successful ELA execution were generated using lakers. Equivalent parameters were aligned in both protocols, e.g. both the anonymous EAP identity and the ELA field LOC W were set to “example.com”, and both EDHOC handshakes sent CRED V by value.

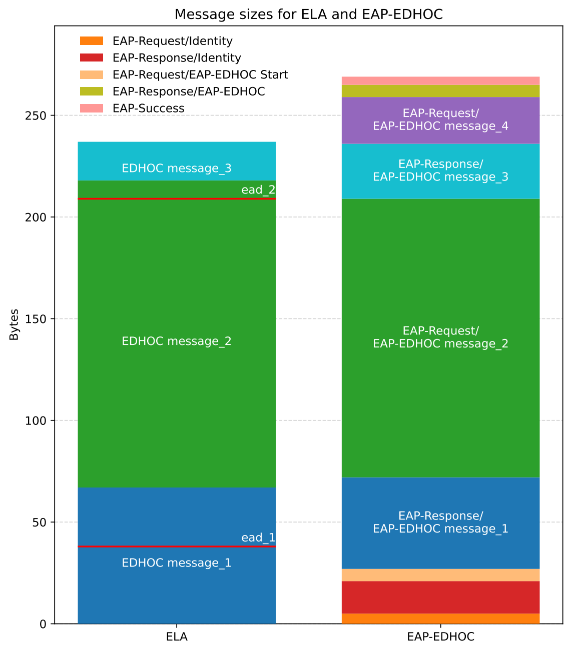


Figure 15. Message sizes for a complete execution of ELA and EAP-EDHOC.

Results, shown in Figure 15, indicate that ELA uses 32 bytes less than EAP-EDHOC. The largest packets in both protocols are EDHOC message 2, which contains a raw public key credential encoded as a CBOR Claims Set (CCS). The EAD1 field uses 43% of message 1 in ELA, while message 4 (optional in ELA) uses 23 additional bytes. Next, we investigate how the 11.8% reduction provided by ELA compares to EAP-EDHOC in terms of execution time, especially when considering the higher message count sent by EAP-EDHOC in the context of low-power and severely duty-cycled IoT networks.

Protocol duration in a duty-cycled network

To measure protocol duration, we played both ELA and EAP-EDHOC over a SmartMeshIP network, where an AIOT Play board served as an EAP peer. Since there is no implementation of EAP-EDHOC available, and given that our main goal is to assess the impacts of using a duty-cycled network, we reused a set of pre-encoded EAP-EDHOC headers, i.e., without implementing a full EAP state machine. Nevertheless, the computations related to EDHOC were executed normally.

Since the SmartMeshIP network formation is non-deterministic, we ran each protocol 100 times to obtain the average execution time. To calculate the duration of each handshake, we synchronized network time, device time, and computer time, and extracted metadata available in packets received at the EAP authenticator.

Results, displayed in Figure 16, indicate that ELA uses 53.63% of the time of EAP-EDHOC for a complete handshake. This confirms that the total overhead size is not necessarily an indicator of the protocol duration in a duty-cycled network. A major contributor is also the packet count. In this case, since the SmartMeshIP network implements a duty cycle, the larger number of messages in EAP-EDHOC causes it to take about double the time when compared to ELA, while the latter transmits only about 1/10 less bytes in total.

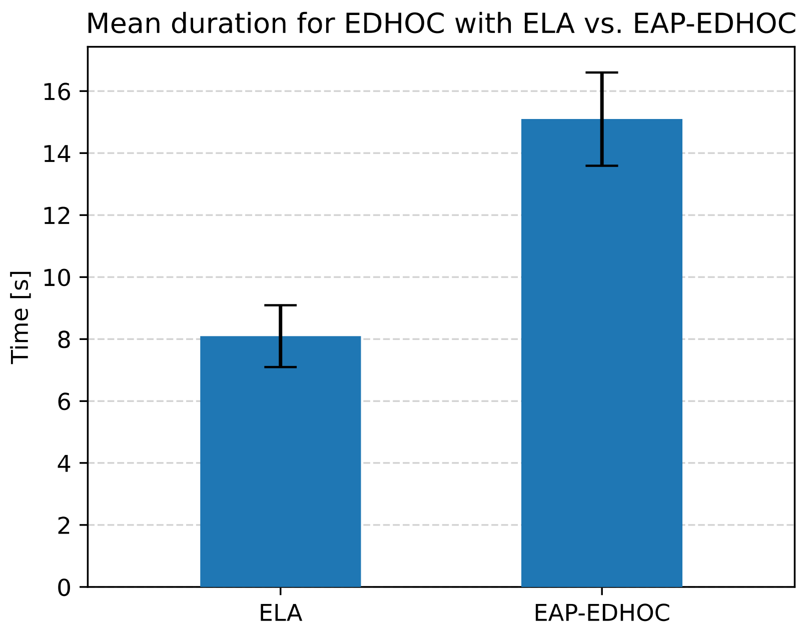


Figure 16. Time taken to complete ELA vs EAP-EDHOC, averaged over 100 runs.

Use Case 2: ELA for the Enrollment of DotBots

Platform

The DotBot (see Fig. 12) is an easy-to-use micro-robot applicable for swarm research [22]. It features an nRF53840 microcontroller, which has a dual core ARM Cortex-M33 processor, 1 MB of flash, and 512 KB of RAM. It communicates using a 2.4 GHz radio, supporting both the IEEE 802.15.4 standard and BLE. The software stack running on

DotBots leverages the raw BLE for communication, meaning that only the physical layer of the standard is used.

A set of DotBots can be controlled by a BLE gateway connected to a controller software that runs on a computer. The gateway can be any BLE-compatible MCU – it acts as a relay, forwarding messages to and from the controller via a serial interface. The controller software is written in Python and handles everything needed for the swarm of DotBots to function, including detecting new robots, calculating indoor positioning [27], and serving a web-based GUI [28].

Integration

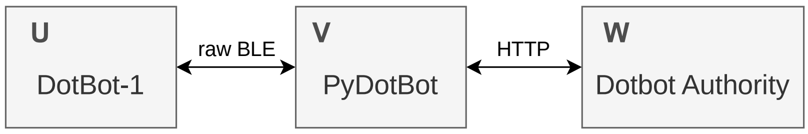


Figure 17. ELA integrated with the DotBot infrastructure.

We use ELA to enroll DotBots, as shown in Figure 17. The DotBot acts as U / EDHOC Initiator: it establishes an EDHOC session with the controller via BLE. The controller implements V / EDHOC Responder: it relays the enrollment request to W to obtain the Voucher. An entity called the DotBot Authority plays the role of the enrollment server (W): it handles the request from V and enforces authorization of the DotBots before emitting the Voucher.

Message Sizes

We measure the message footprint impact of ELA, by executing the protocol and logging the EDHOC messages in V. In particular, we are interested in evaluating the impact of the EAD fields populated by ELA. Figure 18 presents the results: message 1 roughly doubles its size (from 38 to 78 bytes), message 2 has a small overhead (≈ 6%), and message 3 is not affected. It is worth noting that EDHOC message 2 needs to carry the full CRED V by value, which is what enables a true zero-touch enrollment. In our case, CRED V gets encoded to 84 bytes.

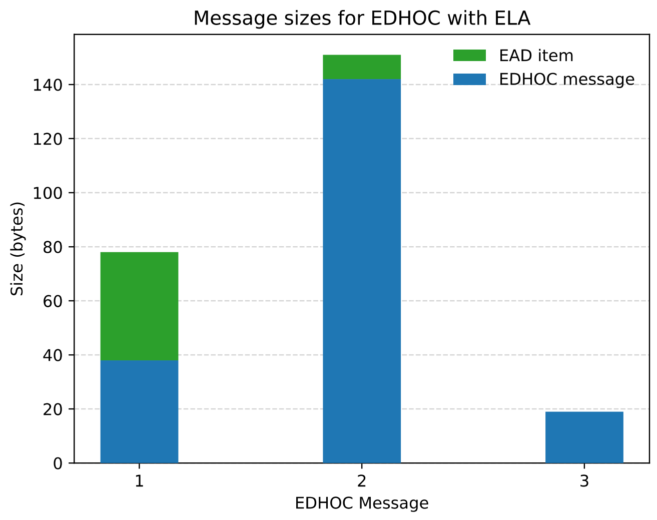


Figure 18. Message sizes for an execution of ELA.

Energy and Execution Time

We measure how long it takes for a DotBot to execute ELA, and how much energy is consumed in each step of the protocol. To do so, we instrumented the DotBot with a power profiler device (Otii Arc) and a logic analyser. The power profiler measures the current drain, while the logic analyser reads from digital pins that are triggered when certain steps of the protocol are executed.

We found that it takes 1.16 s for the DotBot to execute ELA, and during that time it consumes 64.51 mC (see Figure 19). The energy consumption is linear and is only slightly reduced when the DotBot radio is receiving (the radio does not implement duty cycle), while waiting for message 2. The only significant overhead of ELA in the context of the EDHOC handshake is the preparation of EAD1, which takes ≈ 20% (249 ms) of the total handshake time. Using a hardware-based cryptographic accelerator for computation of the Diffie-Hellman shared secret would significantly reduce this time.

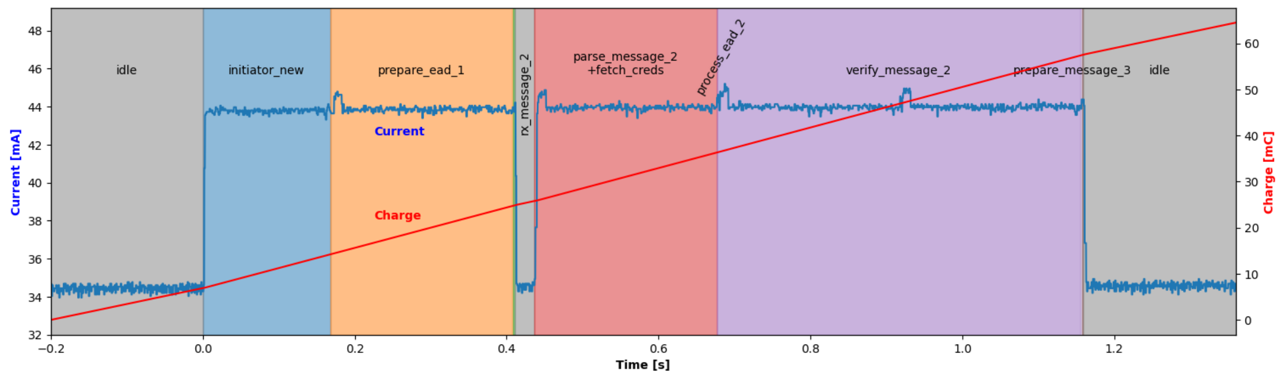


Figure 19. Current-energy benchmark of an execution of ELA on DotBots.

Post-Quantum Security and Hardware Reprogrammability

The protocols described in this Deliverable are generic with respect to the algorithms used in each instance of the protocol. That means that the protocols can be instantiated with different cipher suites that are either already standardized or to-be standardized in the future.

The EDHOC protocol, which serves as the carrier of the zero-touch joining protocol, provides considerations on post-quantum security [8]:

As of the publication of this specification, it is unclear when or even if a quantum computer of sufficient size and power to exploit public key cryptography will exist. Deployments that need to consider risks decades into the future should transition to Post-Quantum Cryptography (PQC) in the not-too-distant future. Many other systems should take a slower wait-and-see approach where PQC is phased in when the quantum threat is more imminent. Current PQC algorithms have limitations compared to Elliptic Curve Cryptography (ECC), and the data sizes would be problematic in many constrained IoT systems.

Symmetric algorithms used in EDHOC, such as SHA-256 and AES-CCM-16-64-128, are practically secure against even large quantum computers. Two of NIST's security levels for quantum-resistant public key cryptography are based on AES-128 and SHA-256. A quantum computer will likely be expensive and slow due to heavy error correction. Grover's algorithm, which is proven to be optimal, cannot effectively be parallelized. It will provide little or no advantage in attacking AES, and AES-128 will remain secure for decades to come [29].

EDHOC supports all signature algorithms defined by COSE, including PQC signature algorithms such as HSS-LMS. EDHOC is currently only specified for use with key exchange algorithms of type ECDH curves, but any Key Encapsulation Method (KEM), including PQC KEMs, can be used in method 0. While the key exchange in method 0 is specified with the terms of the Diffie-Hellman protocol, the key exchange adheres to a KEM interface: G_X is then the public key of the Initiator, G_Y is the encapsulation, and G_XY is the shared secret. Use of PQC KEMs to replace static DH authentication would likely require a specification updating EDHOC with new methods.

While switching a cipher suite in software-based implementations is a matter of a software upgrade, firmware implementations for constrained devices often rely on hardware support for cryptographic acceleration to efficiently execute the necessary algorithms. This means that switching a cipher suite on a constrained device might not be as trivial as updating a firmware image. However, the recent advances in hardware reconfigurable constrained devices due to the use of e.g. FPGAs enable use cases where the hardware support for different cryptographic algorithms is upgradeable, as well. Therefore, we can envision the cases where a hardware upgrade of a constrained device carries new cryptographic accelerators that in effect allow the software implementation of a protocol such as EDHOC to leverage the new cipher suite implementation in hardware.

Conclusion

This document gathers the results achieved in the context of Task 2.2, the development and benchmarking of a zero-touch joining solution. The work done in this task focused on standardizing within the Internet Engineering Task Force (IETF) and researching the performance around this solution. The solution builds on a newly standardized, lightweight protocol for authenticated key exchange called Ephemeral Diffie-Hellman over COSE (EDHOC, RFC 9528 and RFC 9529).

The "zero-touch" is defined as the absence of any manual parameter provisioning on the device being enrolled *at the deployment time*. Instead, when first joining the network before being enrolled, the device talks to its home Enrollment Server, an entity that represents the manufacturer of the device and who authorizes the device to enroll a particular domain.

In this document, we first explain the workflow of the Internet Engineering Task Force (IETF) and how we progressed and standardized the developed solution. We outline our activities related to the organization of a Hackathon on Lightweight IoT Security in Paris on 21-22 May 2024, but also to the general participation in the IETF standardization activities.

Then, we provide the state-of-the-art overview on the different Internet-of-Things enrollment solutions. We can conclude from the existing solutions that they either are technology-specific and/or require a "one touch" of a device at the deployment time. We discuss how the solution proposed in this deliverable fills the current gap on being lightweight, technology-agnostic and zero-touch.

We then introduce EDHOC and OSCORE protocols, whose standardization in the IETF we contributed to, as the base of the "lightweight" facet of the zero-touch joining solution. We describe these protocols and how they contribute to the lightweight IoT security stack.

Building on these protocols, we describe in detail the specification of the zero-touch joining solution, developed in the scope of the IETF standardization: Lightweight Authorization over EDHOC. At the core of the solution is a 3-party protocol that enables two peers to complete the authenticated key exchange without sharing a common root of trust. This is achieved through the help of a third party, the Enrollment Server, which is trusted by the device under enrollment as they share the manufacturer-provided trust anchor. The Enrollment Server provides the device with a Voucher that authorizes it to join and trust a particular domain.

We then proceed to benchmarking against state-of-the-art. First, we benchmark the performance of the EDHOC protocol versus the Datagram Transport Layer Security 1.3 (DTLS) protocol. We show that in the raw public key with mutual authentication configuration over IEEE 802.15.4, the difference in bytes over the air between EDHOC and DTLS is x7.75 in favor of EDHOC. In this same configuration, the handshake duration and energy consumption is improved by x1.88 and x1.93, respectively. We found that flash usage differs in x4.55 and that the total RAM difference is of x3.52 in favor of EDHOC. EDHOC outperformed DTLS in all metrics, with biggest improvements on time-on-air, followed by memory usage, and at last by duration and energy usage.

Second, once we have demonstrated the improvements with respect to the DTLS protocol, we benchmark the zero-touch joining solution. We compare against EAP-EDHOC, an EDHOC-based variant of the Extensible Authentication Protocol (EAP). We demonstrate that even though the total bytes over-the-air difference between ELA and EAP-EDHOC is only 32 bytes in total, because of the chatty nature of EAP (more protocol messages), on average ELA uses 53.33% of the time of EAP-EDHOC to complete the handshake. This confirms the observation that the total overhead size is not necessarily an indicator of the protocol duration in a duty-cycled network: small messages transported over the network have a major influence on the duration, as well. We have also demonstrated how ELA is used for the enrollment of DotBots, a swarm robotic platform used in the context of OpenSwarm.

We have also discussed Post-Quantum considerations of the protocols developed in the context of this deliverable and their cryptographic agility by design.

Bibliography

[1] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy and P. Thubert, "IETF 6TiSCH: A tutorial," *IEEE Communications Surveys & Tutorials,* vol. 22, no. 1, pp. 595-615, 2019.

[2] S. Alvarado-Marin, M. Balbi, A. Abadie, T. Savić, F. Maksimovic and T. Watteyne, "DotBot, a cm-Scale, Easy-to-Use Micro-Robot for Swarm Research," in *Breaking Swarm Stereotypes Workshop at IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[3] LoRa Alliance, *LoRaWAN L2 1.0.4 Specification,* https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf, 2020.

[4] Bluetooth Special Interest Group, *Bluetooth Authorization Control Service 1.0,* https://www.bluetooth.com/specifications/assigned-numbers/authentication-service/, 2020.

[5] Wi-Fi Alliance, *Device Provisioning Protocol Specification,* https://www.wifi.org/downloads-registered-guest/Device_Provisioning_Protocol_Specification_v1.0.pdf, 2018.

[6] K. Wierenga, S. Winter and T. Wolniewicz, "The eduroam Architecture for Network Roaming," *Internet Engineering Task Force,* vol. RFC 7593, 2015.

[7] G. Selander, J. P. Mattsson, F. Palombini and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," *Internet Engineering Task Force,* vol. RFC 8613, 2019.

[8] G. Selander, J. P. Mattsson and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)," *Internet Engineering Task Force,* vol. RFC 9528, March 2024.

[9] IEEE Standards Association, *IEEE 802.15.4-2020: IEEE Standard for Low-Rate Wireless Networks,* 2020.

[10] ZigBee Alliance, *ZigBee Specification,* 2015.

[11] M. Vučinić, J. Simon, K. Pister and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH," *Internet Engineering Task Force,* vol. RFC 9031, 2021.

[12] M. Casar, T. Pawelke, J. Steffan and G. Terhorst, "A survey on Bluetooth Low Energy security and privacy," *Elsevier Computer Networks,* vol. 205, 2022.

[13] IEEE Standards Association, *IEEE Standard for Local and Metropolitan Area Networks---Port-Based Network Access Control,* IEEE Standards Association, 2020.

[14] J. Vollbrecht, J. D. Carlson, L. Blunk, B. . D. Aboba and H. Levkowetz, "Extensible Authentication Protocol (EAP)," *Internet Engineering Task Force,* vol. RFC 3748, 2004.

[15] D. Simon, R. Hurst and B. D. Aboba, "The EAP-TLS Authentication Protocol," *Internet Engineering Task Force,* vol. RFC 5216, 2008.

[16] T. Aura, M. Sethi and A. Peltonen, "Nimble Out-of-Band Authentication for EAP (EAP-NOOB)," *Internet Engineering Task Force,* vol. RFC 9140, 2021.

[17] D. Garcia-Carrillo, R. Marin-Lopez, G. Selander and J. Preuß Mattsson, "Using the Extensible Authentication Protocol with Ephemeral Diffie-Hellman over COSE (EDHOC)," *Internet Engineering Task Force* , Vols. draft-ietf-emu-eap-edhoc-01, 2024.

[18] M. Pritikin, M. C. Richardson, T. Eckert, M. H. Behringer and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)," *Internet Engineering Task Force,* vol. RFC 8995, 2021.

[19] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)," *Internet Engineering Task Force,* vol. RFC 7252, 2014.

[20] J. Preuss Mattsson, G. Selander, S. Raza, J. Höglund and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)," *Internet Engineering Task Force,* Vols. draft-ietf-cose-cbor-encoded-cert-09, 2024.

[21] H. Krawczyk, "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols," in *Annual international cryptology conference.*, Berlin, 2003.

[22] M. Vučinić, G. Selander, J. Preuss Mattsson and T. Watteyne, "Lightweight Authenticated Key Exchange With EDHOC," *IEEE Computer,* 2022.

[23] F. Palombini, M. Tiloca, R. Hoglund, S. Hristozov and G. Selander, "Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)," *Internet Engineering Task Force,* Vols. draft-ietf-core-oscore-edhoc-11, 2024.

[24] M. Rademacher, H. Linka, J. Konrad, T. Horstmann and K. Jonas, "Bounds for the Scalability of TLS over LoRaWAN," in *Mobile Communication Technologies and Applications; 26th ITG-Symposium*, 2022.

[25] H. Ayers, E. Laufer, P. Mure, J. Park, E. Rodelo, T. Rossman, A. Pronin, P. Levis and J. Van Why, "Tighten rust's belt: shrinking embedded Rust binaries," in *Proceedings of the 23rd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2022.

[26] T. Watteyne, L. Doherty, J. Simon and K. Pister, "Technical overview of SmartMesh IP," in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2013.

[27] S. Alvarado-Marin, C. Huidobro-Marin, M. Balbi, T. Savić, T. Watteyne and F. Maksimovic, "Lighthouse Localization of Miniature Wireless Robots," *IEEE Robotics and Automation Letters,* 2024.

[28] A. Abadie, M. Vučinić, D. Badillo, S. Alvarado-Marin, F. Maksimovic and T. Watteyne, "Qrkey: Simply and Securely Controlling Swarm Robots," in *17th International Workshop on Networked Robotics and Communication Systems (NetRobiCS 2024)- IEEE INFOCOM 2024*, 2024.

[29] National Institute Standards and Technology (NIST), "Post-Quantum Cryptography FAQs," [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/faqs.