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

Geovane Fedrecheski, Mališa Vučinić, Thomas Watteyne

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

Geovane Fedrecheski, Mališa Vučinić, Thomas Watteyne
Inria Paris
Paris, France
Email: {first.last}@inria.fr

***Abstract*—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 $\times 6$ to $\times 14$ reduction in packet sizes, $\times 1.44$ improvement in handshake duration and $\times 2.79$ 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 $\times 7$ more time on air than EDHOC. Finally, we measure flash memory and RAM usage, with the EDHOC implementation achieving a $\times 4$ reduction in both.**

***Index Terms*—Internet of Things, Authenticated key exchange, Performance, EDHOC, DTLS.**

I. Introduction

The emergence of the Internet of Things (IoT) has prompted standardization bodies such as the Internet Engineering Task Force (IETF) to develop new protocols that satisfy the constraints imposed by its constrained devices and networks. The results of these efforts span from the lower layers bridging IEEE 802.15.4 and IPv6 through 6TiSCH, extend up to the application layer with the Constrained Application Protocol (CoAP), and address end-to-end security with Object Security for Constrained RESTful Environments (OSCORE). These new specifications tailored for IoT environments address real-world limitations such as low processing power, small batteries, and reduced data rates.

One missing piece in this context consists in providing a lightweight key exchange protocol that can supply session keys to OSCORE. To address this, the Lightweight Authenticated Key Exchange (LAKE) working group in the IETF has recently published the Ephemeral Diffie-Hellman Over COSE (EDHOC) protocol. EDHOC provides a compact handshake and reuses elements already needed by OSCORE, leading to reduced message and code footprints.

As debated in the community, one could argue that Datagram Transport Layer Security (DTLS) already provides the necessary elements for IoT security. In fact, DTLS is the de-facto solution for protecting UDP-based communications, which in turn makes it compatible with CoAP. Furthermore, DTLS has been thoroughly evaluated by the research community and has several available implementations, making it a relevant candidate for authenticated key exchange in the IoT.

Nevertheless, DTLS faces challenges to be used in highly constrained networks, mainly due to its large message footprint. A typical DTLS handshake transfers close to 1 kB of data [1], and performing DTLS handshakes over LoRaWAN networks has been shown to take several minutes [2]. In addition, its relatively complex message flow requires equally complex implementations.

EDHOC [3] allows for lightweight handshakes that typically transfer between 101 and 242 bytes [1], depending on the configuration. Being intentionally developed for IoT environments, it has only three mandatory messages. Also, by reusing elements from OSCORE, code size can be kept low. Finally, EDHOC is transport agnostic and can work even in networks where IP is typically not available, such as LoRaWAN or Bluetooth Low Energy.

In this paper, we evaluate whether the reduced message sizes of EDHOC against DTLS translate into the use of other relevant resources. We implemented both protocols on two typical constrained IoT devices communicating via IEEE 802.15.4 radios. We measured the mean energy and time spent by each handshake, as well as the memory usage. Results show a $\times 7.75$ reduction in message footprint and approximately $\times 1.9$ reduction in energy and time, when comparing EDHOC and DTLS using raw public keys (RPKs) and mutual authentication. We also find that EDHOC uses up to $\times 4$ less flash and RAM than DTLS.

We also provide a simulation of LoRaWAN time on air. We computed the message sizes for EDHOC and DTLS, and computed the time on air that these messages would require. Results show that, in the worst case (LoRa spreading factor = 12), EDHOC takes almost 7 seconds to transmit, while the DTLS configurations take up to 50 and 80 seconds.

The main contribution of this paper is a comparison of the formally published version of EDHOC (draft-22) and DTLS 1.3 across several metrics (message footprint, time, energy, memory) using real hardware and under constrained

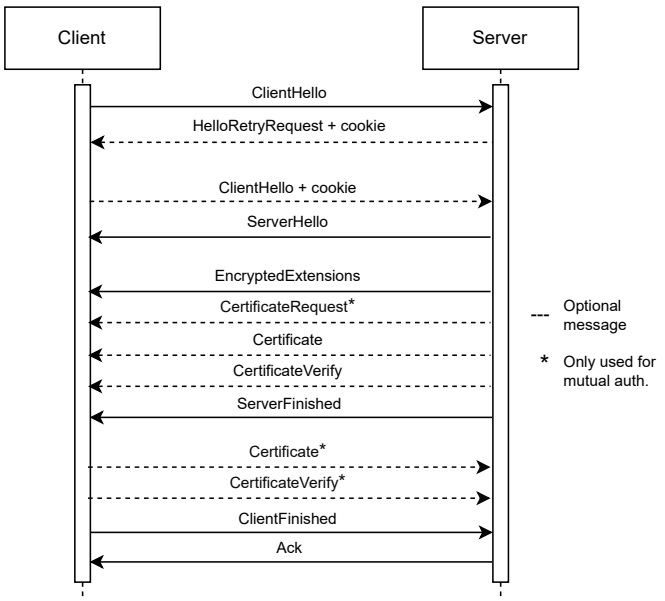


Fig. 1: The DTLS 1.3 handshake. Our evaluation includes all the optional messages.

IoT networks (IEEE 802.15.4 and LoRaWAN). To the best of our knowledge, we are the first to show how the commonly measured message footprint correlates with handshake duration, energy consumption, and memory usage. We release the code and data used in this paper as open source[1].

The rest of this paper is organized as follows. Section II provides an overview of DTLS and EDHOC. Section III presents the evaluation, including the experimental setup and the results. We discuss related work in Section IV and present a discussion of our findings in Section V. Finally, Section VI concludes the paper.

II. Background

A. DTLS 1.3

The Transport Layer Security (TLS) protocol, standardized by the IETF and currently at version 1.3 [4], provides state-of-the-art authentication, integrity, and confidentiality to messages exchanged over the Internet. TLS was defined to work over a reliable transport channel, most commonly Transmission Control Protocol (TCP).

To enable similar security properties in connectionless transports, the IETF developed the Datagram TLS (DTLS) [5]. DTLS, also currently at version 1.3, is defined as a delta of TLS, which has the benefit of enabling code reuse. Among the main differences to TLS are the presence of an explicit counter (to enable out-of-order delivery) and a mechanism to counter Denial of Service (DoS) attacks based on a cookie exchange during the handshake.

In this paper we evaluate the handshake of DTLS 1.3. We show the DTLS 1.3 message flow in Fig. 1, with optional messages in dashed lines. The handshake begins with the client and server exchanging ClientHello and ServerHello messages,

[1]https://github.com/geonnave/edhoc-dtls-comparison

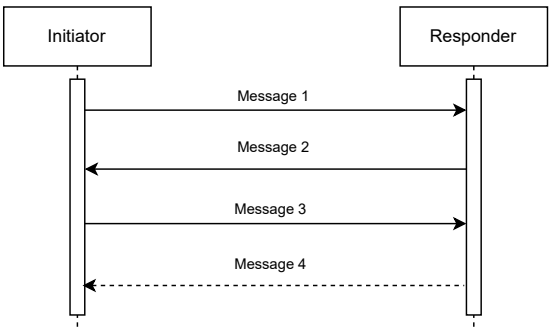


Fig. 2: The EDHOC handshake. The last message is optional. Our evaluation includes the three first messages and a final CoAP acknowledgment.

respectively. These have the goal of establishing shared keying material and selecting the cryptographic parameters. To counter DoS attacks, DTLS uses an optional cookie-based mechanism that involves re-sending the initial messages.

Next, beginning with EncryptedExtensions, all messages are encrypted. Three optional messages follow, in which the server asks the client for its certificate, sends its own certificate, and sends a signature over the handshake content so far. After the ServerFinished message, the client may also send its certificate and certificate verification signature. The handshake finishes with a ClientFinished message followed by an acknowledgment (Ack) from the server. At this point the client and server are ready to communicate securely.

Entities using DTLS can choose to authenticate based on pre-shared keys, certificates, or raw public keys (RPKs), which must be signaled during the handshake. DTLS also optionally supports certificate verification with a chain of trust. The security requirements and capabilities of the participating entities determine how these parameters are selected. For example, pre-shared keys are computationally cheaper and result in smaller message sizes when compared to RPKs or certificates. Similarly, RPK-based authentication uses less bytes over the air than certificates, but do not provide root of trust verification.

Since the main motivation for DTLS was to support operation in connectionless transports, it does not necessarily satisfy requirements typically found in IoT environments. For example, a DTLS handshake with RPKs transmits a total of 894 bytes of data [1], which can be challenging to send in low-power IoT networks. In addition, the several messages exchanged during a DTLS handshake require more elaborate state machines, leading to implementation complexity and potentially larger memory requirements.

B. EDHOC

The Lightweight Authenticated Key Exchange working group of the Internet Engineering Task Force (IETF) has developed the Ephemeral Diffie-Hellman Over COSE (EDHOC) protocol [3]. Its goal is to enable mutual authentication and the derivation of secret keys between two constrained parties, the initiator and the responder, with messages small

enough to be used in low-power IoT networks. A typical EDHOC handshake consists of three messages of sizes 37, 45, and 19 bytes. EDHOC is agnostic of the transport layer and can be used in environments without IP, although its recommended and default setting is to be carried over the reliable CoAP. The main use case of EDHOC is to supply session keys to OSCORE, which in turn provides end-to-end message encryption.

Devices using EDHOC can authenticate by using either X.509 certificates or raw public keys as credentials. To minimize message sizes, EDHOC may only send the credential identifiers, in which case the application is responsible for resolving the actual credentials. Authentication is achieved via static Diffie-Hellman or signature keys, including any combination of them.

We illustrate the EDHOC handshake in Fig. 2: it consists of three messages, plus an optional fourth message. In the first message, the initiator sends its intended authentication method and cipher suite, as well as an ephemeral public key. The responder then sends back its own ephemeral key, along with an encrypted credential identifier and an authentication item (either a signature or a MAC). In both messages, connection identifiers are also exchanged.

The third message is encrypted and authenticated by the derived secret, and it carries the initiator's credential identifier as well as an authentication item. Upon receiving it, the responder derives the shared secret and optionally sends the fourth message. Next, the shared secret can be used to protect application messages, e.g., via the OSCORE protocol [6].

One of the goals of the EDHOC specification is implementation simplicity [7]. To achieve that, it reuses several elements already needed by OSCORE, such as CBOR, COSE, and CoAP. In addition, it uses a simple model for message flights, consisting of only three mandatory messages. Furthermore, several low-cost optimizations are used in order to reduce message sizes, such as the use of CBOR sequences and compact representation of string identifiers [3].

One of the main goals of EDHOC is to allow secure key establishment in severely constrained IoT networks, and it achieves that by being very efficient with respect to message footprint. It remains an open question, however, whether this reduction in message footprint translates to a reduction of other important resources in IoT environments, such as processing time, energy, and memory. Similarly, it remains unknown how these metrics fare when compared to DTLS 1.3, arguably EDHOC's most relevant alternative.

III. 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.

A. 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 [8], an IoT operating system, and its `gnrc` network implementation.

To evaluate EDHOC, we use the `edhoc_rs` [2] Rust crate, linked as a static library in RIOT. To evaluate DTLS, we use `wolfSSL` [3], 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 Fig. 3. We can see that EDHOC sends $\times 6.09$ and $\times 14.35$ 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 I. Thus, the MAC headers are sent many times over the network, further accentuating the message sizes differences. Table I also shows the accumulated bytes over the air per

[2]https://github.com/openwsn-berkeley/edhoc-rs/

[3]https://www.wolfssl.com/

layer of the network stack. Note that 6LoWPAN contains the compressed IPv6 and UDP headers.

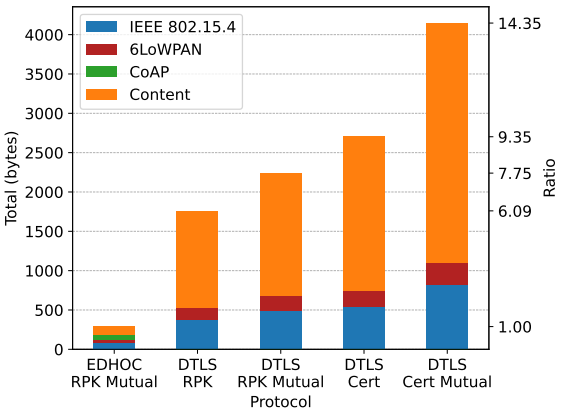


Fig. 3: Bytes over the air for EDHOC and DTLS handshakes.

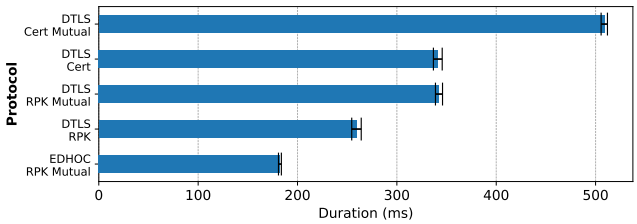
TABLE I: Messages and fragments per handshake, and the amount of bytes over the air for each layer.

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

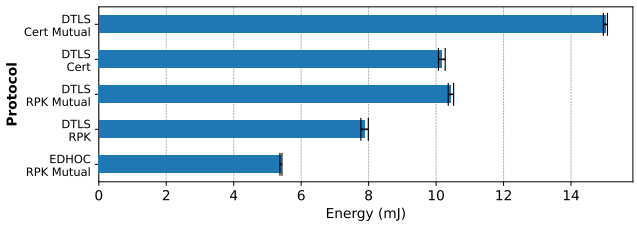
Handshake duration and energy consumption: Next, we look into the question of how the difference in bytes over the air translates into the usage of other resources: handshake duration and energy consumption. We measure it using a power profiler device (Otii Arc[4]) connected to the device that initiates 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.

Fig. 4 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 $\times 1.44$ more time and energy compared to EDHOC. The DTLS configuration with certificates and mutual authentication uses $\times 2.79$ 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 packet sizes reduction is due to the compact serialization used by EDHOC. Small packets 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



(a) Handshake Duration



(b) Energy Consumption

Fig. 4: Results of time and energy measurements for a handshake under several configurations.

notably the generation of ephemeral asymmetric key pairs and performing a Diffie-Hellman operation.

B. 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 [9], 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 [2], 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[5] to estimate the time on air for EDHOC and DTLS over LoRaWAN in several spreading factor configurations.

[4]https://www.qoitech.com/otii-arc-pro/

[5]https://github.com/tanupoo/lorawan_toa

Figure 5 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.

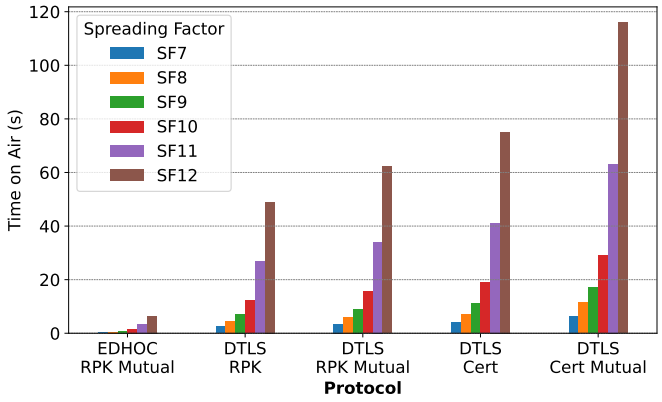
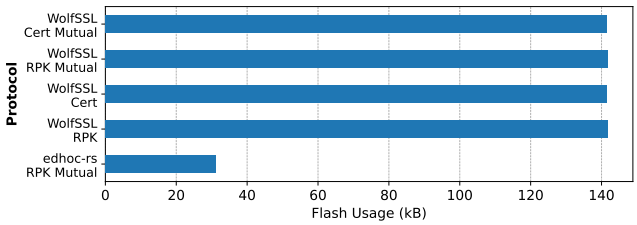
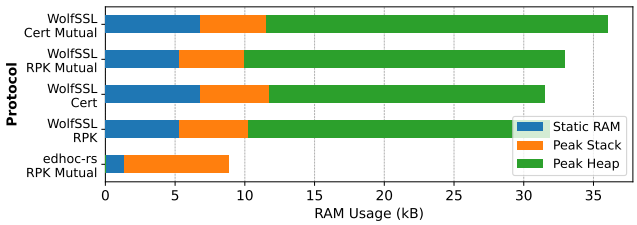


Fig. 5: Time on air for EDHOC and DTLS handshakes under different spreading factors (not considering duty cycle).



(a) Flash memory



(b) RAM.

Fig. 6: Flash memory and RAM usage for the `edhoc_rs` and `wolfSSL` implementations.

C. Memory Usage

We measure memory usage in two steps. First, we measure flash memory and static RAM by analysing the `.map` files generated during compilation. In this analysis we only include the symbols relevant to the library (either `edhoc_rs` or `WolfSSL`) 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[6] to obtain maximum heap usage. Note that peak heap is only measured for `WolfSSL`, since `edhoc_rs` does not use dynamic memory.

Flash memory: Fig. 6 presents the RAM and flash footprint results. `wolfSSL` uses virtually the same amount of flash in all configurations, and requires about $\times 4.5$ more flash than `edhoc_rs`. 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 `edhoc_rs` implementation uses only inline CBOR processing, i.e., it does not depend on a CBOR library. It is also worth noting that the `edhoc_rs` 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, `edhoc_rs` is written in Rust, which typically leads to binaries twice as large than C [10].

Volatile memory: Fig. 6b presents the results on RAM footprint. Overall, `WolfSSL` uses between $\times 3.6$ and $\times 4$ more RAM than `edhoc_rs`. The `edhoc_rs` 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 I).

[6]Memory painting consists in filling the memory with a known pattern and scanning it after some processing to assess how much of it has changed.

IV. Related Work

Several works assess the amount of resources used by Internet-of-Things security protocols. In Table II, we provide a summary of the previous literature, designating which works evaluate DTLS, EDHOC, or both, as well as the evaluated version. Most works assess either DTLS or EDHOC, with two exceptions Durand *et al.* [11] and Mattsson *et al.* [1], that compare both protocols.

Table II also compares our work to others in terms of which metrics were evaluated for the selected protocols. All works evaluate message sizes, while the metrics of time, energy, and memory are evaluated by a count of 4, 2, and 2 works, respectively. Our work compares the latest version of DTLS and EDHOC, as published by the IETF, across all four metrics.

TABLE II: Related work. **V** and **D** indicate the DTLS **V**ersion and EDHOC **D**raft number; **L** and **I** indicate **L**oRaWAN and **I**EEE 802.15.4.

Work	DTLS	EDHOC	Time	Energy	Msg. Size	Memory	Network
[12]	V-1.2	D-06	✓		✓		L
[13]		D-08	✓		✓		I
[11]	V-1.3	D-12	✓	✓	✓		
[14]	V-1.3			✓	✓	✓	I
[15]		D-02	✓		✓	✓	
[2]	V-1.2		✓		✓		L
[1]	V-1.3	D-18			✓		
Ours	V-1.3	D-22	✓	✓	✓	✓	I, L

Among the works that use LoRaWAN, Sanchez-Iborra *et al.* [12] use EDHOC to derive network and application keys, while Rademache *et al.* [2] measure ToA

for TLS and find significant bottlenecks, especially in the downlink.

From the point of view of DTLS, a comparison of versions 1.2 and 1.3 concludes that the latter adds a small overhead in flash and RAM [14]. The same study observes that there is a relatively small difference in bytes-over-the air with DTLS compared to TLS. Several works compare DTLS and EDHOC with respect to bytes-over-the-air [1], [11], [14], and conclude that exchanged data is significantly reduced when using the latter. Others propose modifications to EDHOC to reduce even more the amount of transmitted data [13].

Another work proposes and evaluates two C libraries for OSCORE and EDHOC respectively, and evaluates them in different IoT hardware [15]. Both libraries use a relatively small amount of flash ($<= 19kB$) and integrate key management with trusted execution environments. Among the differences to our work are the choice of language (`edhoc_rs` is written in Rust), that their evaluation does not use cryptographic hardware accelerators, and that the measurements are done in the bare metal configuration, i.e., without a network stack.

V. Discussion

As shown in this and previous works, EDHOC offers a notable reduction in the bytes-over-the-air footprint. Since it implies less use of the radio, it also saves energy. We note that the smaller sizes are achieved at the cost of operational flexibility, since the evaluated EDHOC handshakes only send credential references. This means that a credential provisioning procedure is needed when EDHOC is used.

We observed that EDHOC reduces up to $\times 2.79$ the time and energy needed to complete a handshake. Shorter handshake time means less latency from the application point of view, which could be noticeable by users [16]. Note that these metrics may be impacted by the chosen authentication method, cipher suite, and cryptographic backend.

We noted a significant improvement of EDHOC over DTLS in the memory usage. While this result is implementation-dependent, we have shown that both Flash and RAM usage in `edhoc_rs` correspond to around $\times 4$ of the figures used by `wolfSSL`. Among the implications are a potential cost reduction, and having more memory made available for the application code.

VI. Conclusion

In this paper, we compare EDHOC, a newly standardized key exchange protocol for IoT with (D)TLS 1.3, the de-facto Internet security solution. Previous works identified that EDHOC offers smaller message footprints when compared to DTLS. We investigate how this reduction in message sizes translates to the use of other scarce resources in IoT environments, including time, energy, and memory. We evaluate EDHOC and DTLS 1.3 implementations on a constrained device using the IEEE 802.15.4 radio, and by simulating time-on-air for LoRaWAN networks.

We show that, in the *RPK with mutual authentication* configuration over IEEE 802.15.4, the difference in bytes over the air is of $\times 7.75$ in favor of EDHOC. In this same configuration, the handshake duration and energy consumption is improved by $\times 1.88$ and $\times 1.93$, respectively. Next, we found that flash usage differs in $\times 4.55$ and that the total RAM difference is of $\times 3.52$. Similarly, in the LoRa simulation, we learned that with $SF = 12$ there is a time-on-air improvement of $\times 9.69$. Therefore, 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.

References

[1] J. P. Mattsson, F. Palombini, and M. Vučinić, "Comparison of CoAP Security Protocols," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-lwig-security-protocol-comparison-07, January 2023.

[2] 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*. VDE, 2022, pp. 1–6.

[3] G. Selander, J. P. Mattsson, and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)," Internet Engineering Task Force, Internet-Draft draft-ietf-lake-edhoc-22, Aug. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc/22/

[4] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[5] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[6] G. Selander, J. P. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," RFC 8613, Jul. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8613

[7] M. Vučinić, G. Selander, J. P. Mattsson, and T. Watteyne, "Lightweight authenticated key exchange with EDHOC," *Computer*, vol. 55, no. 4, pp. 94–100, 2022.

[8] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 79–80.

[9] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and J.-C. Zúñiga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," RFC 8724, Apr. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8724

[10] 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, pp. 121–132.

[11] A. Durand, P. Gremaud, J. Pasquier, and U. Gerber, "Trusted lightweight communication for IoT systems using hardware security," in *Proceedings of the 9th International Conference on the Internet of Things*, 2019, pp. 1–4.

[12] R. Sanchez-Iborra, J. Sánchez-Gómez, S. Pérez, P. J. Fernández, J. Santa, J. L. Hernández-Ramos, and A. F. Skarmeta, "Enhancing lorawan security through a lightweight and authenticated key management approach," *Sensors*, vol. 18, no. 6, p. 1833, 2018.

[13] S. Pérez, J. L. Hernández-Ramos, S. Raza, and A. Skarmeta, "Application layer key establishment for end-to-end security in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2117–2128, 2019.

[14] G. Restuccia, H. Tschofenig, and E. Baccelli, "Low-power IoT communication security: On the performance of DTLS and TLS 1.3," in *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*. IEEE, 2020, pp. 1–6.

[15] S. Hristozov, M. Huber, L. Xu, J. Fietz, M. Liess, and G. Sigl, "The cost of OSCORE and EDHOC for constrained devices," in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, 2021, pp. 245–250.

[16] J. Deber, R. Jota, C. Forlines, and D. Wigdor, "How much faster is fast enough? user perception of latency & latency improvements in direct and indirect touch," in *Proceedings of the 33rd annual acm conference on human factors in computing systems*, 2015, pp. 1827–1836.