Abstract

The Internet Engineering Task Force (IETF) and its Lightweight Authenticated Key Exchange working group have produced a solution that enables public-key based authenticated key exchange over the most constrained Internet of Things radio communication technologies. We describe the Ephemeral Diffie-Hellman over COSE (EDHOC) protocol, its expected security properties, and invite the community for a formal study.

Lightweight Authenticated Key Exchange with EDHOC

Mališa Vučinić
Inria, Paris
malisa.vucinic@inria.fr

Göran Selander
Ericsson Research
goran.selander@ericsson.com

John Preuß Mattsson
Ericsson Research
john.mattsson@ericsson.com

Thomas Watteyne
Inria, Paris
thomas.watteyne@inria.fr

January 19, 2022

Keywords: IETF, EDHOC, 6TiSCH, LoRAWAN, NB-IoT

Introduction

Low-power radio communication technologies like 6TiSCH [1], LoRAWAN [2] or NB-IoT enable multi-year lifetime of Internet-of-Things (IoT) devices even when they are powered with off-the-shelf batteries. They also present inherent communication challenges: bandwidth is scarce and can be as low as byte / second; communication may be intermittent and prone to several-second delays; maximum transmission units are on the order of 50 bytes. Devices that support these networks are typically equipped with micro-controllers running at 10's of MHz, with 10's of kB of Random Access Memory (RAM) and several hundred kB of code memory. They have some level of hardware acceleration of cryptographic algorithms: AES acceleration is commonly present, acceleration of operations and algorithms over different elliptic curves (e.g. NIST P-256, Curve25519), and hash functions (e.g. SHA-256) is becoming increasingly present in new chip designs.

The Internet community has focused during this past decade on enabling these IoT devices and networks to run Internet Protocol (IP)-based communication stacks. The Internet Engineering Task Force (IETF) first standardized an IPv6 compression layer called 6LoWPAN in 2007, followed by the standardization of new protocols that are adapted for low-power networks. The Constrained Application Protocol (CoAP) was published in 2014 as the specialized web transfer protocol. Its security extension, Object Security for Constrained RESTful Environments (OSCORE) requires an efficient key exchange protocol.

The IETF formed the Lightweight Authenticated Key Exchange (LAKE) working group in 2019; it collected requirements and started working on a solution. A deliverable of the standardization effort, the Ephemeral Diffie-Hellman over COSE (EDHOC) protocol, is expected to become an important component in securing constrained networks and devices of the Internet of Things (IoT). EDHOC will be submitted for publication as an Internet Standard (RFC) in 2022. The working group solicits formal analysis of the latest version of the protocol [3] by the community to incorporate feedback before it is submitted for publication. The goal of this document is to summarize the relevant aspects of the protocol to facilitate formal analysis.

Motivation and Use Cases

EDHOC is designed to enable public-key based authenticated key exchange of two peers potentially running on constrained devices over low-power IoT radio communication technologies. The keys derived by EDHOC can be used to protect the applications, which includes the protection of data through authenticated encryption. One example application is the OSCORE protocol standardized in RFC8613 [4]. The three main use cases considered by the working group for EDHOC are *(1)* IETF 6TiSCH, multihop mesh networking technology; *(2)* LoRaWAN, low-power wide area technology; *(3)* NB-IoT, low-power cellular technology. To illustrate the challenge of performing authenticated key exchange over these technologies, consider that a typical LoRaWAN packet is 51 bytes long and takes 2.8 s of airtime before a next packet can be sent, assuming the frequently used LoRA SF12 spreading factor and the 125 kHz bandwidth. Other technologies have different but similarly constraining requirements on performance, which has influenced the protocol design with the overarching goals:

- minimize the number of messages to complete the protocol;
- minimize message sizes to reduce the number of fragments;
- minimize code and memory footprint by reusing primitives that are already used by security-related protocols in other parts of the protocol stack. These include CBOR [5] for efficient encoding and COSE [6] for object security.

As an example, one instance of the EDHOC protocol has three messages of sizes 37, 45, and 19 bytes.

Internet Threat Model

We consider the traditional Internet threat model, as documented in RFC3552 [7]. The communicating endpoints are trustworthy and the attacker has nearly complete control over the communication channel. The attacker can read, remove, change, or inject forged messages.

Security Goals

We summarize here the security goals of the protocol.

Mutual Authentication. At the end of the protocol session, each peer shall have freshly authenticated the other peer's long-term credentials. Peers shall agree on a fresh session identifier, roles and credentials of both peers.

Confidentiality. Only the two peers authenticated during the protocol session shall be in possession of the derived shared secret. By compromising the long-term credential of either peer, an attacker shall not be able to compute past session keys (forward secrecy).

Downgrade Protection. The protocol should account for potentially long deployment times by including modular and negotiable support for cryptographic primitives. At the end of the protocol session, both peers shall agree on both the cryptographic algorithms that were proposed and those that were chosen.

Security Level. The protocol should establish a key with a target security level of ≥ 127 bits.

Identity Protection. The protocol should protect the identity of one peer against active attackers, and the identity of the other peer against passive attackers.

Protection of External Data. The protocol should allow for external security applications to piggyback data within specific fields in the protocol messages. The external data shall have the same level of protection as the protocol message it is carried within. In EDHOC, this data is called External Authorization Data (EAD) and is carried in each protocol message.

Protocol Design

Building Blocks

The cryptographic core of EDHOC is based on the theoretical SIGMA-I protocol through its MAC-then-Sign variant [8], complemented by the key schedule inspired by the Noise `XX` pattern [9]. Compact encoding is achieved through the use of CBOR, standardized in RFC8949 [5]. EDHOC uses the cryptographic algorithms standardized in COSE in RFC8152 [6], a wrapper around different key derivation functions instantiated based on the selected hash function, with a custom definition of a binary additive stream cipher for unauthenticated encryption leveraging the key derivation Expand function. EDHOC is not bound to a particular transport layer, although it is expected to be mainly transported over CoAP. EDHOC relies on the transport layer to handle message loss, message reordering, message duplication, fragmentation, demultiplexing, Denial-of-Service (DoS) protection against non-routable addresses, and message correlation.

Overview

The two EDHOC peers are denoted as the Initiator and the Responder. The key exchange in EDHOC is based on ephemeral Elliptic Curve Diffie-Hellman keys. Each peer authenticates using its long-term credential, which can either be a signature key or a static Diffie-Hellman key. The type of credential of each peer determines the authentication method (see Table 1), which is agreed out-of-band.

When a peer uses a signature key for authentication, the protocol message sent by that peer includes a signature. When a peer uses a static Diffie-Hellman key to authenticate, the signature is replaced by a (shorter) message authentication code (MAC). The MAC is in fact calculated as an output of the Extract-and-Expand functions keyed with the Diffie-Hellman static-ephemeral secret. We illustrate how the different keys and MACs are derived in Fig. 2.

The Initiator selects a cipher suite for the session. It generates an ephemeral Diffie-Hellman key X and signals G_X to the Responder. If the Responder accepted the cipher suite, it continues the protocol and sends `message_2` to the Initiator, which includes the Responder's ephemeral Diffie-Hellman key. The Initiator verifies `message_2` and continues the protocol by sending `message_3`.

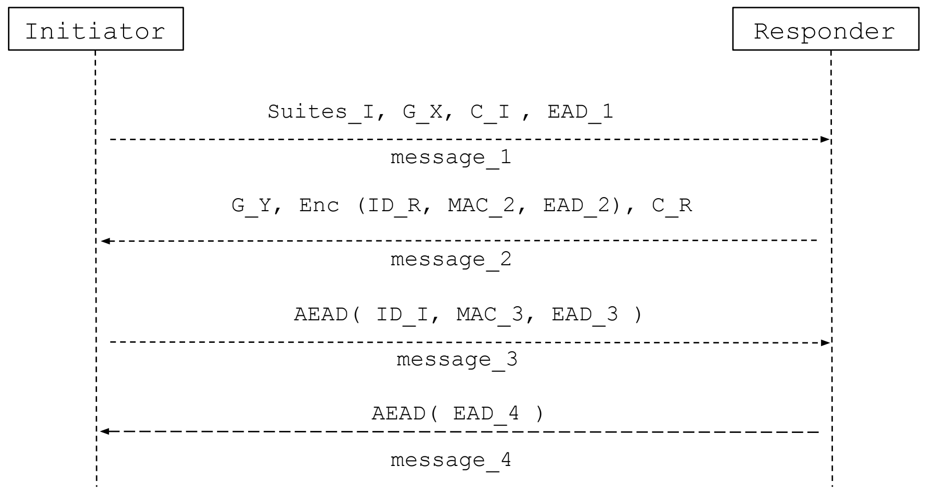
Initiator
Responder
Suites_I, G_X, C_I , EAD_1
message_1
G_Y, Enc (ID_R, MAC_2, EAD_2), C_R
message_2
AEAD(ID_I, MAC_3, EAD_3)
message_3
AEAD(EAD_4)
message_4

Figure 1: The EDHOC protocol when instantiated with `STAT-STAT` method. The generic protocol is defined in `draft-ietf-lake-edhoc-12` [3]. Enc() denotes unauthenticated encryption, while AEAD() denotes authenticated encryption with associated data. MAC denotes "message authentication code", and is calculated as an output of the Expand function (see Fig. 2). EAD stands for External Authorization Data.

Once the Responder receives and successfully verifies `message_3`, peers are mutually authenticated and in possession of a shared session secret. At this point, the protocol specifies an optional `message_4` which the Responder can use to explicitly signal the key confirmation. Depending on the use case, the key confirmation can also be achieved by sending the application data protected with the exported EDHOC key.

Table 1: Authentication Method Types.

Id	Initiator	Responder	Abbreviation
0	Signature	Signature	`SIG-SIG`
1	Signature	Static DH	`SIG-STAT`
2	Static DH	Signature	`STAT-SIG`
3	Static DH	Static DH	`STAT-STAT`

Cipher Suites and Extensibility

EDHOC defines the concept of a cipher suite as an ordered set of standardized algorithms and a MAC length parameter used for static DH authentication methods.

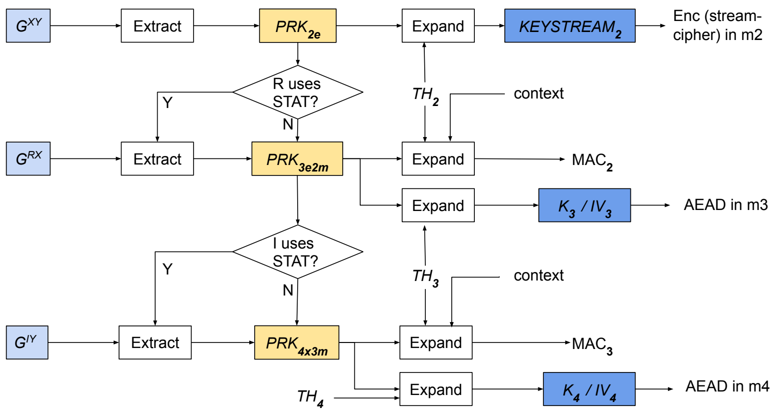

Figure 2: EDHOC key schedule, adapted from Norrman *et al.* [10]. Light blue boxes denote Diffie-Hellman shared secrets (G^{XY}, G^{RX}, G^{IY}), where X and Y denote the ephemeral keys, and I and R the long-term static DH keys. Yellow boxes denote intermediate keying material (PRK_{2e}, PRK_{3e2m}, PRK_{4x3m}). Dark blue boxes denote AEAD keys and Initialization Vectors (IVs), and the keystream for the binary additive stream cipher used for encryption in `message_2`. Conditional boxes forward the input to the output, depending on the evaluation of the condition (Y/N). TH_i denotes transcript hashes.

In principle, any combination of standardized algorithms is possible, but the specification pre-defines several suites which are deemed efficient for constrained devices. These include combinations of AES-CCM authenticated encryption with different tag lengths, the underlying elliptic curves (P-256, Curve25519) and signature algorithms (ECDSA, EdDSA). The predefined cipher suites are listed in Table 3. Additionally, through cipher suite values that explicitly denote private use, the specification allows deployment-specific combinations of algorithms.

EDHOC supports all the signature algorithms and authentication credential types defined by COSE [6]. A COSE signature is determined by the signature algorithm and the authentication key algorithm, just like in TLS 1.3 and IKEv2. The authentication key algorithm depends on the type of the authentication credential (signature or static DH).

Post-Quantum resistance is not the main focus of the protocol as it targets constrained environments. However, the EDHOC method `SIG-SIG` supports Post-Quantum Cryptography (PQC) signatures: the key exchange in `SIG-SIG` can trivially be exchanged with a PQC Key Encapsulation Method (KEM). The Ephemeral-Static ECDH implicit authentication cannot trivially be replaced with a PQC KEM. EDHOC with KEMs for authentication would require the standardization of a new method.

Generic Key Derivation Function

Table 2: Instantiating EDHOC Extract and Expand functions.

Hash algorithm	Extract()	Expand()
SHA-2	HKDF-Extract	HKDF-Expand
SHAKE128	KMAC128	KMAC128
SHAKE256	KMAC256	KMAC256

EDHOC uses a generic Extract-and-Expand key derivation function which is instantiated based on the hash algorithm in the selected cipher suite (see Table 2). The intent is to align the hash algorithm used in the cipher suite with the one in the key derivation function, and so require a single hash implementation on a constrained device.

Key Schedule

EDHOC derives a fixed-length uniformly pseudorandom key (PRK) from the DH shared secrets using the Extract function. Method `SIG-SIG` derives a single PRK (PRK_{2e}) using the ephemeral-ephemeral shared secret. The theoretical SIGMA-I protocol does not specify any specific key schedule. EDHOC uses a key schedule inspired by the noise protocol where a new key is derived for each use and each derivation use as much available information as possible. Static DH methods derive the PRKs from all the available DH shared secrets (ephemeral-ephemeral, static-ephemeral for each peer). Each time a shared secret is available, it is passed to the Extract function together with the previous PRK. The PRK at different stages of the protocol, the transcript hashes and the context information are passed as an input to the Expand function to derive the intermediary keying material: a keystream for the binary additive stream cipher for the encryption operation of `message_2`, MAC keys in `message_2` and `message_3`, and AEAD keys in `message_3` and `message_4`. Fig. 2 illustrates the key schedule. Method `STAT-STAT` is similar to the Noise `XX` protocol but uses the MAC-then-Encrypt approach from SIGMA instead of the Encrypt-then-MAC approach used in Noise `XX`.

Derivation of Application Keys

`EDHOC-Exporter` function allows applications to derive the keying material based on the performed protocol session. As an input to the function, each application provides a static unique label value that is assigned through the standardization process, together with a runtime specific context parameter and the desired key length. The application keying material is an output of the Expand function with PRK_{4x3m} of the EDHOC session as the pseudorandom keying material,

Table 3: Pre-defined EDHOC cipher suites. The ECDSA signature algorithm is standardized in RFC8152 as ES256 or ES384, depending on the use of SHA-256 or SHA-384 as a hash function, respectively.

Id	AEAD	Hash	MAC len	ECDH curve	Signature	Application AEAD	Note
0	AES-CCM-16-64-128	SHA-256	8	X25519	EdDSA	AES-CCM-16-64-128	constrained
1	AES-CCM-16-128-128	SHA-256	16	X25519	EdDSA	AES-CCM-16-64-128	constrained
2	AES-CCM-16-64-128	SHA-256	8	P-256	ES256	AES-CCM-16-64-128	constrained
3	AES-CCM-16-128-128	SHA-256	16	P-256	ES256	AES-CCM-16-64-128	constrained
4	ChaCha20/Poly1305	SHA-256	16	X25519	EdDSA	ChaCha20/Poly1305	
5	ChaCha20/Poly1305	SHA-256	16	P-256	ES256	ChaCha20/Poly1305	
6	A128GCM	SHA-256	16	X25519	ES256	A128GCM	
24	A256GCM	SHA-384	16	P-384	ES384	A256GCM	high-security
25	ChaCha20/Poly1305	SHAKE256	16	X448	EdDSA	ChaCha20/Poly1305	high-security

transcript hash TH_4 and application inputs as the additional `info` parameter. Fig. 3 illustrates `EDHOC-Exporter`.

Lightweight Rekeying

EDHOC provides an `EDHOC-KeyUpdate` function for applications to perform lightweight rekeying without needing to rerun the complete protocol. The input to the function is a nonce (e.g. a counter or a random number) that needs to be provided by the application and agreed upon by the Initiator and the Responder. The nonce and the PRK_{4x3m} of the current session are passed as inputs to the Extract function. The output of Extract replaces the PRK_{4x3m} session keying material. The function aims at providing forward secrecy: the compromise of a long-term authentication key does not compromise past session keys, and the compromise of a session key does not compromise past session keys. However, the compromise of a single session key does lead to the compromise of all future session keys derived using the `EDHOC-KeyUpdate` function. In use cases where this is not desirable, it is necessary to re-run the complete EDHOC protocol. Fig. 4 illustrates `EDHOC-KeyUpdate`.

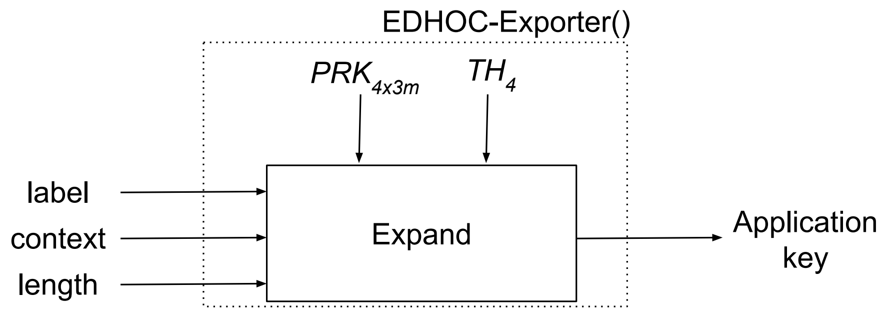

Figure 3: `EDHOC-Exporter` function is used for the derivation of application keying material. Label, context and length are provided by the application. Label is a static value registered for each EDHOC application.

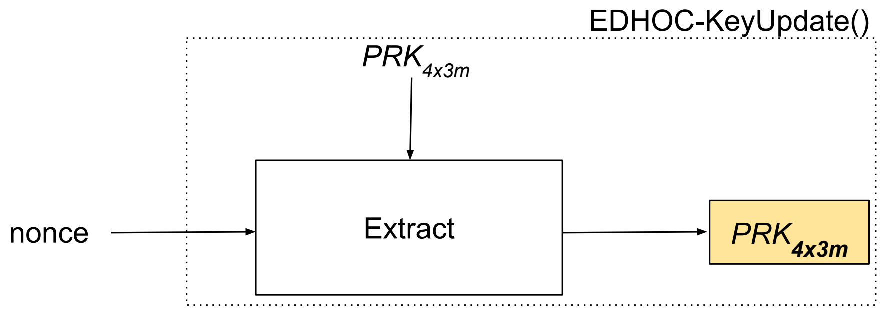

Figure 4: `EDHOC-KeyUpdate` function. Nonce is provided by the application.

Discussion

How EDHOC Meets the Security Goals

We discuss here how the EDHOC design attempts to meet the security goals presented before. We also reference the relevant proofs available for earlier versions of the protocol. Note that a thorough discussion of the expected EDHOC properties is presented in the Security Considerations section in the specification [3].

Mutual authentication. Norrman *et al.* [10] studied the `-00` version of the specification in the symbolic Dolev-Yao model with idealized cryptographic primitives using Tamarin. They prove the mutual injective agreement property that covers the identity of the Responder, roles of the peers, session keying material, connection identifiers and cipher suites. In case the Initiator is using static DH keys, the proof does not cover the Initiator identity and the Initiator's ephemeral-static DH key share. The injective agreement proof for the Responder covers both of these parameters. They additionally prove implicit agreement for both the Initiator and the Responder. Based on the injective and implicit agreement properties, the authors infer the KCI resistance.

Confidentiality. Norrman *et al.* [10] prove the forward secrecy of the session keying material for all authentication methods. In addition, the specification defines the `EDHOC-KeyUpdate` function for lightweight rekeying. The output of the Extract function in `EDHOC-KeyUpdate` replaces the current session keying material (PRK_{4x3m}). We intuitively argue that this construction provides forward secrecy.

Downgrade protection. The injective agreement proof by Norrman *et al.* [10] covers the cipher suites and connection identifiers. We therefore argue that the protocol is resistant against downgrade attacks.

Identity Protection. EDHOC makes the same design tradeoffs as TLS 1.3 and IKEv2, and opts for the SIGMA-I variant of the SIGMA protocol: the Responder's identity is protected against passive attackers and the Initiator's identity is protected against active attackers. These properties are expected to hold also for Static DH-based authentication methods, an extension to the original SIGMA design.

Protection of External Authorization Data (EAD). EAD_1 and EAD_2 transported in `message_1` and `message_2`, respectively, are considered unprotected by EDHOC. EAD_3 and EAD_4 are considered protected. We note in the specification that the external security applications using these fields need to avoid including sensitive information, which may break the security properties of the protocol.

Non-repudiation was discussed as an additional property that is useful for LAKE use cases, but has not be set as a goal. If either EDHOC peers authenticates with a signature, the other peer can prove that it performed a protocol session by presenting the input to the signature function as well as the signature itself. With both peers using static DH keys, both peers can deny having participated in the protocol session.

Challenges

EDHOC used with the authentication method `SIG-SIG` is an instance of the MAC-then-Sign variant of the SIGMA-I protocol. Other EDHOC methods use static DH keys for authentication, an aspect that departs from the traditional SIGMA design. EDHOC's method `STAT-STAT` is an abstraction of SIGMA with both SIGMA signatures replaced by MACs calculated from the ephemeral-static DH shared secrets. While this is a similar approach as used by Noise `XX`, EDHOC `STAT-STAT` is not a direct instance of Noise `XX`: EDHOC `STAT-STAT` uses MAC-then-Encrypt approach to align with SIGMA-I, instead of Encrypt-then-MAC used by Noise `XX`. EDHOC keeps the MAC-then-Encrypt SIGMA structure also in `SIG-STAT` and `STAT-SIG` methods with heterogeneous authentication keys, but replaces the signature with a MAC calculated from an ephemeral-static DH shared secret whenever a peer uses a static DH key. In summary, the building blocks of these static DH methods originate from solid and formally verified protocols. *The research question we pose to the community is whether the integration of SIGMA and Noise `XX`, as used in EDHOC, is secure in the computational model and whether the protocol meets the security goals outlined before? Have the changes introduced in the protocol since it underwent the symbolic analysis introduced a regression?*

Another important aspect to consider is the *security level*. As a reminder, the message size is an inherent constraint of LAKE environments. EDHOC specifies cipher suites that in case of static DH keys allow for MACs of at least 64-bits. Methods `SIG-STAT` and `STAT-SIG` use a signature in one direction and a MAC in the other direction. Method `STAT-STAT` does not use a signature operation at all. *The questions we pose to the research community is whether the security level goal is met, taking into account all the possible combinations of EDHOC methods, cipher suites and algorithms?*

Finally, real-world protocols are as secure as their implementations. At the time of the writing, EDHOC has been implemented through seven independent implementations. These include general purpose Java and Python-based implementations, as well as the implementations in (memory unsafe) C targeting embedded systems. The implementations were tested for interoperability through several interop testing events. To increase confidence in the security of the implementations targeting embedded systems, we invite the community to also study and contribute the executable code.

Acknowledgment

The authors would like to thank Karthik Bhargavan and Stephen Farrell for the provided feedback, as well as the participants of the IETF LAKE working group for their input during the standardization process.

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, doi: 10.1109/COMST.2019.2939407.

[2] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, “Understanding the limits of lorawan,” *IEEE Communications magazine*, vol. 55, no. 9, pp. 34–40, 2017, doi: 10.1109/MCOM.2017.1600613.

[3] G. Selander, J. Mattsson, and F. Palombini, *Ephemeral Diffie-Hellman Over COSE (EDHOC)*, Internet-Draft, work in progress, IETF Std. draft-ietf-lake-edhoc-12, October 2021.

[4] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, *Object Security for Constrained RESTful Environments (OSCORE)*, IETF Std. RFC8613, July 2019.

[5] C. Bormann and P. Hoffman, *Concise Binary Object Representation (CBOR)*, IETF Std. RFC8949, July 2019.

[6] J. Schaad, *CBOR Object Signing and Encryption (COSE)*, IETF Std. RFC8152, July 2017.

[7] E. Rescorla, B. Korver, and I. A. Board, *Guidelines for Writing RFC Text on Security Considerations*, IETF Std. RFC3552, July 2003.

[8] H. Krawczyk, “SIGMA: The ‘SIGn-and-MAc’ approach to authenticated Diffie-Hellman and its use in the IKE protocols,” in *Annual International Cryptology Conference.* Springer, 2003, pp. 400–425, doi: 10.1007/978-3-540-45146-4_24.

[9] T. Perrin. (2018) The Noise Protocol Framework. Revision 34. Accessed: 12 January 2022. [Online]. Available: http://noiseprotocol.org/noise.html

[10] K. Norrman, V. Sundararajan, and A. Bruni, “Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices,” in *International Conference on Security and Cryptography (SECRYPT)*, 2021, pp. 210–221, 10.5220/0010554002100221.