Lightweight Authenticated Key Exchange with EDHOC: Design Overview

Mališa Vučinić, Göran Selander, John Preuss Mattsson

Lightweight Authenticated Key Exchange with EDHOC: Design Overview

Mališa Vučinić, Göran Selander, John Preuß Mattsson

***Abstract*—The Internet Engineering Task Force (IETF) and its Lightweight Authenticated Key Exchange (LAKE) working group have produced a key exchange solution that enables public-key -based key exchange over the most constrained Internet of Things (IoT) radio communication technologies. The EDHOC protocol, a deliverable of this standardization effort, prioritizes small message size to minimize the number of radio frames. As an example, one instance of the protocol completes the authenticated key exchange of two peers with three messages of 37, 45 and 19 bytes. The protocol will be submitted for publication as an Internet Standard (RFC) and expects a wide deployment on constrained devices, with 7 independent implementations already available. It is substantial that the expected security properties hold. We invite the formal analysis community to study the protocol and contribute the results in both the symbolic and the computational model. The feedback from the community will be incorporated into the specification before it is published. The goal of this document is to summarize the EDHOC design, the expected properties and to outline the open research questions, in order to facilitate the analysis.**

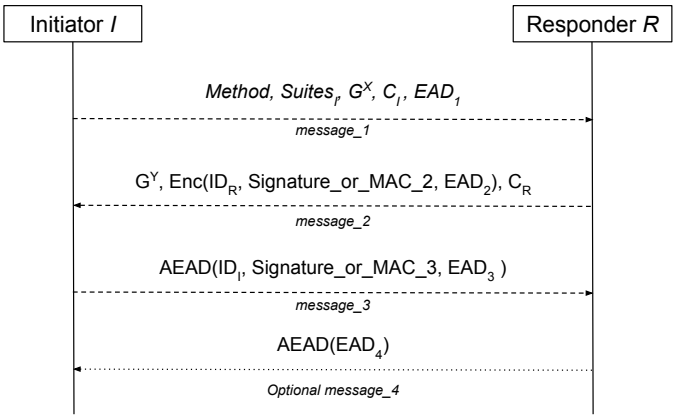

Fig. 1. EDHOC draft-12 protocol, which defines Elliptic Curve Diffie-Hellman key exchange. Enc() denotes unauthenticated encryption, AEAD() denotes authenticated encryption with associated data. EAD stands for External Authorization Data.

I. Context

The Lightweight Authenticated Key Exchange (LAKE)[1] working group within the Internet Engineering Task Force (IETF) is nearing the completion of its main work item [1]. 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). The working group solicits formal analysis of the latest version of the protocol [2] by the community to incorporate the feedback before its publication. The goal of this document[2] is to summarize the relevant aspects of the protocol to facilitate formal analysis.

II. 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 for 3^{rd} party applications, including the protection of application data through authenticated encryption. One example application is the OSCORE protocol standardized in RFC 8613. The three main use cases considered by the working group for EDHOC are 1) LoRaWAN low-power wide area technology; 2) IETF 6TiSCH, multihop mesh networking technology; and 3) NB-IoT, a 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 consumes 2.8 seconds of airtime before a next packet can be sent[3]. Other technologies have different but similarly constraining requirements on performance, which has influenced the protocol design with the overarching goals to minimize:

- Number of messages to complete the protocol;
- Message sizes to minimize the number of radio fragments when protocol is transported over the considered radio technologies;
- Code and memory footprint through the reuse of primitives that are already present on constrained devices.

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

EDHOC is expected to be deployed on Class 1 and Class 2 constrained devices based on microcontrollers (see RFC 7228), as well as on more powerful, general purpose machines. Constrained devices often 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, Ed25519),

M. Vučinić is with Inria, France.
G. Selander and J. Preuß Mattsson are with Ericsson Research, Sweden

[1] https://datatracker.ietf.org/wg/lake/about/

[2] The authors made every attempt to ensure correctness of this document. However, in case of any inconsistencies with the IETF specification(s), the IETF documents have precedence.

[3] Assumes the frequently used LoRA SF12 spreading factor and the 125 kHz bandwidth. See https://www.thethingsnetwork.org/airtime-calculator.

and hash functions (e.g. SHA-256) is becoming increasingly present in new chip designs.

III. Internet Threat Model

We consider the traditional Internet threat model, as documented in RFC 3552. The communicating endpoints are trustworthy and the attacker has a nearly complete control over the communication channel. The attacker can read, remove, change, or inject forged messages.

IV. Security Goals

We summarize here the security goals of the protocol, as discussed in [1].

A. Mutual Authentication.

At the end of the protocol session, each peer shall have freshly authenticated the other peer's long-term credential. Peers shall agree on a fresh session identifier, roles and credentials of both peers. This includes the following properties:

- Key Compromise Impersonation (KCI) resistance.
- Identity Misbinding resistance. Note that the identity may be related directly to the public key (public key or hash of the public key as the identity), or may be the data unrelated to the key.
- Reflection resistance.

B. 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).

C. 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.

D. Security Level

The protocol should establish a key with a target security level of $\geq$ 127 bits.

E. Identity Protection

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

F. 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.

V. Protocol Design

A. Building Blocks

The cryptographic core of EDHOC is based on the theoretical SIGMA-I protocol through its MAC-then-Sign variant [3], and complemented with the key schedule inspired by the Noise `XX` pattern [4]. Compact encoding is achieved through the use of CBOR, standardized in RFC 8949. EDHOC uses the cryptographic algorithms standardized in COSE [4], a wrapper around different key derivation functions instantiated based on the selected hash function, and 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.

B. Overview

EDHOC peers are denoted as the Initiator (I) and the Responder (R). The key exchange in EDHOC is based on ephemeral 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 I), signaled in the first EDHOC message (see Fig. 1). When a peer uses a signature key for authentication, the protocol message sent by that peer includes a signature, following the SIGMA design. When a peer uses a static DH key to authenticate, the signature in the SIGMA design is replaced by a (shorter) message authentication code (MAC), which is calculated based on the ephemeral-static DH shared secret. Protocol fields `Signature_or_MAC_x` are simply populated by either a signature or a MAC, depending on the exact authentication method in use.

The Initiator selects a cipher suite for the session, signals it to the Responder together with more preferred supported cipher suites (if any) as part of $\texttt{Suites}_I$ in the first message. The Responder discontinues the session by triggering an error message if it does not support the selected cipher suite or if it supports another cipher suite in $\texttt{Suites}_I$. Subsequently, the Initiator selects a different cipher suite for the following protocol session. Fig. 1 depicts the protocol assuming the Responder accepted the selected cipher suite. 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[5]. Depending on the use case, the key confirmation can also be achieved by sending the application data protected with the exported EDHOC key.

[4]The full list of algorithms is available at https://www.iana.org/assignments/cose/cose.xhtml.

[5]The forth message was added as a result of symbolic model analysis using Tamarin [5].

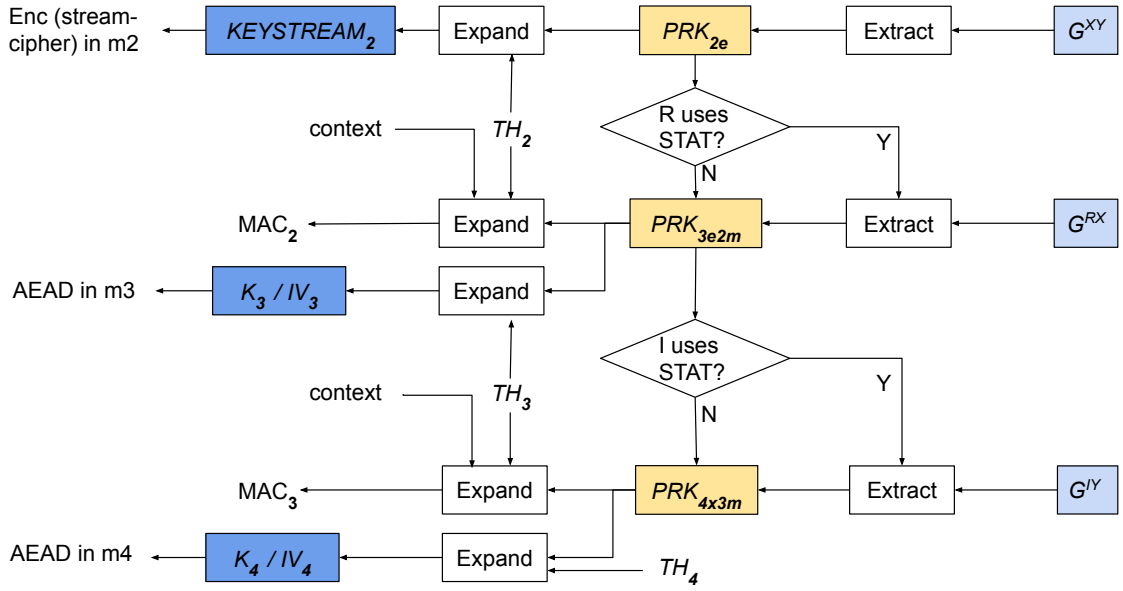

Fig. 2. EDHOC key schedule. Light blue boxes denote Diffie-Hellman shared secrets (G_{XY}, G_{RX}, G_{IY}), where X and Y denote the ephemeral keys, and I and R the long-term static DH keys. Yellow boxes denote intermediate keying material (PRK_{2e}, PRK_{3e2m}, PRK_{4x3m}). Dark blue boxes denote AEAD keys and Initialization Vectors (IVs), and the keystream for the binary additive stream cipher used for encryption in message_2. TH_i denotes transcript hashes. Adapted from [5].

TABLE I
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`

TABLE II
PARAMETERS OF AN EDHOC CIPHER SUITE.

Parameter	Possible values	Note
AEAD algorithm	Table VI	
Hash algorithm	Table VII	
MAC length	$\geq$ 8 bytes	for Static DH
Key exchange algorithm	Table VIII	an ECDH curve
Signature algorithm	Table IX	for Methods 0-2
Application AEAD algorithm	Table VI	
Application hash algorithm	Table VII	

C. 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. A given cipher suite is identified with an integer, which encodes the parameters listed in Table II.

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[6] with different tag lengths, the underlying elliptic curves (Ed25519, P-256) and signature algorithms (EdDSA, ECDSA standardized as ES256). The predefined cipher suites are listed in Table IV. 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). The authentication algorithm parameters defined for COSE are listed in Table X. In addition to these, COSE signature algorithms can also be used with X.509 authentication keys.

Post-Quantum resistance is not the main focus of the protocol as it targets constrained environments. However, note that 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.

D. Generic Key Derivation Function

TABLE III
INSTANTIATING EDHOC EXTRACT AND EXPAND FUNCTIONS.

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

[6]AES-CCM is widely implemented in hardware for constrained devices.

TABLE IV
PRE-DEFINED EDHOC CIPHER SUITES.

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

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 III). 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.

E. 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. 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. Transcript hashes (TH_2, TH_3, TH_4) do not make use of a running hash to reduce memory requirements of the implementations.

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

G. Lightweight Rekeying

EDHOC provides a `EDHOC-KeyUpdate` function for applications to perform lightweight rekeying without needing to rerun the complete protocol. The input to the function is a nonce 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`.

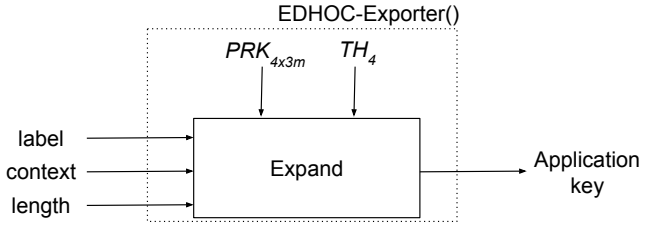

Fig. 3. `EDHOC-Exporter` function is used for the derivation of application keying material. Label, context and length are provided by the application. Label is a static value registered for each EDHOC application.

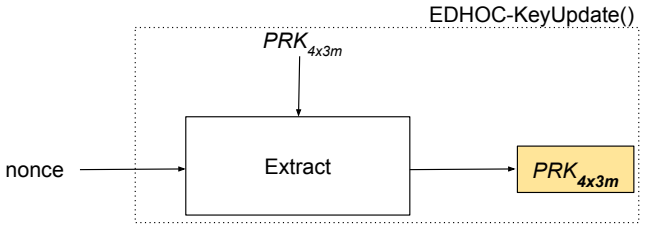

Fig. 4. `EDHOC-KeyUpdate` function. Nonce is provided by the application.

H. Comparison of EDHOC `SIG-SIG` with TLS 1.3 Handshake

Similarly to EDHOC, both IKEv2 and TLS 1.3 are based on SIGMA. EDHOC's method `SIG-SIG` with signature keys is similar to TLS 1.3 handshake, with the following notable considerations:

- EDHOC uses MAC-then-Sign (like IKEv2), TLS 1.3 uses Sign-then-MAC[7].

[7]The SIGMA paper describes both variants, as well as Sign-and-MAC, as secure.

- EDHOC does not provide extra randomness in addition to the randomness in the ephemeral public keys[8]. Compared to TLS 1.3 this lowers the complexity against theoretic pre-computation attacks.
- EDHOC uses the same AEAD scheme as TLS 1.3 with a pseudo-random Initialization Vector (IV) that is kept secret.
- EDHOC uses 64 – 512 bit MACs. TLS handshake use 256-512 bit + 8-16 bit MACs. TLS 1.3 uses 64-128 bit MACs for record layer data origin authentication.
- Protocol aspects such as cipher suite negotiation, bidding-down protection, reflection, and identity misbinding protection are different.
- Transcript hashes and key derivation are different.
- EDHOC includes a nonce in the `EDHOC-KeyUpdate` function to avoid short cycles and bind it to the application round-trip that triggered the `EDHOC-KeyUpdate`.

VI. Discussion

A. How EDHOC Meets the Security Goals

We discuss here how the EDHOC design attempts to meet the security goals presented in Section IV. 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 [2]. Table V summarizes the properties previously studied, as well as the security goals outlined in Section IV.

Mutual authentication. Norrman *et al.* [5] analyzed 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.[9]. 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.* [5] 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 (see Section V-G). 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.* [5] covers the cipher suites and connection identifiers. We therefore argue that the protocol is resistant against downgrade attacks.

Security level. Through the usage of different standardized algorithms and tag lengths, the EDHOC design aims at offering 128-bit security against offline brute force attacks and 64-bit security against online brute force attacks[10].

Identity Protection. EDHOC makes the same design trade-offs 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.

B. 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 results of the symbolic model analysis [5] confirm our intuition. *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 in Section IV? 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

[8]The security proof in SIGMA is without extra randomness.

[9]This is expected as EDHOC aims at providing the identity protection for the Initiator through the SIGMA-I protocol core.

[10]To break 64-bit security against online brute force, an attacker would on average have to send 4.3 billion messages per second for 68 years, which is considered as infeasible in constrained IoT radio technologies.

TABLE V
CANVAS OF SECURITY GOALS AND PROPERTIES. A PROPERTY THAT HAS ALREADY BEEN PROVED FOR AN OLDER VERSION OF THE SPECIFICATION USING SYMBOLIC ANALYSIS IS DENOTED WITH Ⓢ. A PROPERTY THAT IS CLAIMED IN THE SPECIFICATION BUT HAS NOT BEEN FORMALLY PROVED IN EITHER THE SYMBOLIC OR THE COMPUTATIONAL MODEL IS DENOTED WITH ◯. A PROPERTY THAT IS NOT CLAIMED IS DENOTED WITH ×.

		SIG-SIG	SIG-STAT	STAT-SIG	STAT-STAT	Note
Mutual authentication	Injective agreement (I)	Ⓢ	Ⓢ	Ⓢ*	Ⓢ*	See [5]. *Does not cover identity of I and ephemeral-static DH key share of I.
	Injective agreement (R)	Ⓢ	Ⓢ	Ⓢ	Ⓢ	See [5].
	Implicit agreement (mutual)	Ⓢ	Ⓢ	Ⓢ	Ⓢ	See [5].
	KCI resistance	Ⓢ	Ⓢ	Ⓢ	Ⓢ	Inferred, see [5].
	Identity misbinding (Unknown key share)	Ⓢ	Ⓢ	Ⓢ	Ⓢ	Inferred, see [5].
	Reflection resistance	◯	◯	◯	◯	
Confidentiality	Forward secrecy of session keys	Ⓢ	Ⓢ	Ⓢ	Ⓢ	See [5].
	Forward secrecy of EDHOC-KeyUpdate	◯	◯	◯	◯	
Downgrade protection		◯	◯	◯	◯	Injective agreement proof covers connection IDs and cipher suites [5].
Security level (bits)		$\geq$ 64* $\geq$ 128§	$\geq$ 64* $\geq$ 128§	$\geq$ 64* $\geq$ 128§	$\geq$ 64* $\geq$ 128§	*against online attacks, §against offline (claimed).
Identity protection		◯	◯	◯	◯	I against active, R against passive attacks.
Protection of external data		◯	◯	◯	◯	EAD_1 and EAD_2 are not claimed protected, EAD_3 and EAD_4 are claimed.
	Non-repudiation	◯	◯*	◯*	×	*Claimed for the peer authenticating with a signature.

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 design aims at a 64-bit security level against online attacks. *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 7 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.

VII. CONCLUSION

This document summarizes the design of the EDHOC protocol, a deliverable of the standardization process at IETF LAKE. EDHOC enables public-key -based authenticated key exchange over low-power constrained IoT radio communication technologies. EDHOC will be submitted for publication as an Internet Standard (RFC). We solicit from the community the formal analysis of the protocol and will incorporate the received feedback in the standard specification.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] M. Vučinić, G. Selander, J. Mattsson, and D. Garcia-Carillo, *Requirements for a Lightweight AKE for OSCORE*, Internet-Draft, IETF Std. draft-ietf-lake-reqs-04, June 2020.
[2] 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, Oct 2021.
[3] 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.
[4] T. Perrin, "The noise protocol framework," Revision 34, 2018. [Online]. Available: http://noiseprotocol.org/noise.html
[5] K. Norrman, V. Sundararajan, and A. Bruni, "Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices," in *Proceedings of the 18th International Conference on Security and Cryptography (2021)*, 2021, pp. 210–221.
[6] J. Schaad, *CBOR Object Signing and Encryption (COSE)*, IETF Std. RFC 8152, July 2017.

TABLE VI
EDHOC AEAD ALGORITHMS.

Algorithm	Tag length (bits)	Key length (bits)
AES-CCM-16-64-128	64	128
AES-CCM-64-64-128	64	128
AES-CCM-16-128-128	128	128
AES-CCM-64-128-128	128	128
AES-CCM-16-64-256	64	256
AES-CCM-64-64-256	64	256
AES-CCM-16-128-256	128	256
AES-CCM-64-128-256	128	256
A128GCM	128	128
A192GCM	128	192
A256GCM	128	256
ChaCha20/Poly1305	128	256

TABLE VII
EDHOC HASH ALGORITHMS.

Algorithm	Digest length (bits)
SHA-256	256
SHA-384	384
SHA-512	512
SHA-512/256	256
SHAKE128	256 (in COSE)
SHAKE256	512 (in COSE)

TABLE VIII
EDHOC ECDH CURVES.

Algorithm	Security level (bits)
P-256	128
P-384	192
P-521	256
X25519	128
X448	224

TABLE IX
EDHOC SIGNATURE ALGORITHMS (SECURITY LEVEL DEPENDS ON AUTHENTICATION KEY ALGORITHM).

Algorithm	Security level (bits)
ES256	$\leq$ 128
ES512	$\leq$ 256
ES384	$\leq$ 192
EdDSA	128 or 224
PS256	$\leq$ 128
PS384	$\leq$ 192
PS512	$\leq$ 256
HSS-LMS (PQC)	$\geq$ 128

TABLE X
ECC AUTHENTICATION KEY ALGORITHM PARAMETERS (COSE). SECURITY LEVEL DEPENDS ON THE SIGNATURE ALGORITHM OR THE MAC LENGTH IN CASE OF STATIC DH.

Algorithm	Authentication method	Security level (bits)
P-256	Static DH	64–128
P-384	Static DH	64–192
P-512	Static DH	64–256
P-256	Signature (ES256, ES384, or ES512)	128
P-384	Signature (ES256, ES384, or ES512)	128 or 192
P-512	Signature (ES256, ES384, or ES512)	128, 192 or 256
X25519	Static DH	64–128
X448	Static DH	64–224
Ed25519	Signature (EdDSA)	128
Ed448	Signature (EdDSA)	224
RSA	Signature (PS256, PS384, or PS512)	$\geq$ 112
HSS-LMS	Signature (HSS-LMS)	$\geq$ 128