Fine-grained, privacy-augmenting LI-compliance in the LAKE standard

Pascal Lafourcade[1], Elsa López Pérez[2], Charles Olivier-Anclin[1], Cristina Onete[3], Clément Papon[3], and Mališa Vučinić[2]

[1] LIMOS, Université Clermont Auvergne, France
[2] Inria, France
[3] XLIM, Université de Limoges, France

Abstract. The Internet Engineering Task Force and its LAKE working group standardized the Ephemeral Diffie-Hellman over COSE (EDHOC) authenticated key-exchange protocol for use in constrained Internet of Things deployments. The use cases include cellular networks, such as NB-IoT, but also non-cellular networks such as 6TiSCH, and LoRaWAN. As a result of its use in cellular networks, EDHOC will be subject to Lawful Interception (LI), which allows a group of authorities to break, if equipped with a warrant, the end-to-end (E2E) security of the channel established through EDHOC. Current implementations of EDHOC would only allow lawful interception by using the cellular network operator as a legitimate endpoint, essentially running a Person-in-the-Middle attack against the protocol. In this work, we focus on a privacy-preserving, fine-grained LI-compliant modification of EDHOC for all four authentication methods that this protocol currently supports. We achieve this via a careful white-box composition of EDHOC with the Lawful Interception Key-Exchange approach of Arfaoui et al. (ESORICS 2021) and Bultel and Onete (SAC 2022). Our resulting construction not only achieves strong key-security, but also non-frameability, and LI-compliance, without breaking the identity-protection property of EDHOC. Our implementation results show that, while LIKE adds an overhead to a standard EDHOC implementation in Rust, the resulting protocol remains practical while achieving much better privacy and LI-compliance.

1 Introduction

Privacy is a human right. The Universal Declaration for Human Rights [4], states: "*No one shall be subjected to arbitrary interference with his privacy [...] or correspondence [...]. Everyone has the right to the protection of the law against such [...] attacks.*" The United Nations describes privacy as a cornerstone of other basic human rights, like "*the free development and expression of an individual's personality, identity, and beliefs, and their ability to participate in political, economic, social and cultural life*" [5].

[4] `https://www.un.org/en/universal-declaration-human-rights/`
[5] UNO, `https://www.ohchr.org/en/special-procedures/`

Edward Snowden's 2013 revelations of widespread mass surveillance engendered public outcry. Advocates for sacrificing privacy for the sake of (inter-)national security clashed with privacy supporters warning of censorship and autocracy. Examples of illegal mass-surveillance abound today. The NSA once collected call data from all Verizon customers, and data pertaining to calls in the Bahamas and Afghanistan [11]. During the COVID-19 pandemic, contact-tracing data was used in order to facilitate criminal investigations [10]. A recently-proposed EU regulation on moderating child sexual abuse material (CSAM) in encrypted communications enables mass client-side scanning, in spite of out-spread criticism from both scientists [24] and socio-economical entities [6].

Mass-surveillance is a threat to basic human rights. Yet, even strong privacy advocates agree that investigations *limited in scope and motivation* (by lawfully-obtained warrants) can be legitimate [1]. This is the type of limited Lawful Interception (LI) that the *EU resolution on security through encryption and security despite encryption* supports [9].

Lawful Interception (LI) has been a legal and technical requirement for over 30 years in mobile communications, featuring prominently in 3GPP specifications [6]. Every operator that provides end-to-end encrypted user communication is subject to such requirements.

LAKE. The Lightweight Authenticated Key Exchange (LAKE) working group of the Internet Engineering Task Force (IETF) standardized a lightweight secure-channel establishment for use in NB-IoT, 6TiSCH, and LoRaWAN. Since it will be used in mobile environments, LAKE must comply with LI.

The solution proposed by LAKE is EDHOC [25]: a mutually-authenticated lightweight secure-channel establishment scheme, guaranteeing identity-protection as a form of privacy. The Initiator and Responder may choose to authenticate either through signatures or by using static DH-keys. As described in Section 3, the protocol completes in 1.5, optionally 2, round trips. Starting from the 2nd message, communications may be authenticated and/or encrypted using intermediate secrets.

EDHOC is not LI-compliant by design. Naïvely, this can be achieved by turning mobile infrastructure nodes into endpoints – as is the case for other protocols, like AKA; this comes, however, at the cost of an *unnecessary loss of privacy*, making mobile network operators complicit to mass surveillance.

In this paper, we aim to render the EDHOC protocol LI-compliant without this massive loss of privacy. Ideally, our LI-EDHOC scheme must guarantee the security of exchanged messages except with respect to their sender, the receiver, and the collaborative efforts of *all* the Lawful Interception authorities[7]permitted

[6] See technical specifications TS33126 ; TS33127 ; TS33128

[7] LI is performed differently from one country to another. For instance in France, Lawful Interception requires the technical cooperation of the operator, the legislative branch, and law-enforcement. We model each participant that requires a cryptographic key as an authority – which allows us to construct our protocol in an elegant manner. In other countries, potentially more authorities might be required. Our system is flexible and can address all such scenarios. Our protocol is designed

to intercept. Moreover, interception should be fine-grained, limited only to one session at a time.

We use the pairing-free Lawful-Interception Key-Exchange (LIKE) approach [5] described in Section 2. Applying LIKE to a real-world protocol such as EDHOC, which features four methods of authentication and complex key schedules, while also preserving EDHOC's lightweight character and identity-protection, is far from trivial. An important challenge, for instance, stems from the fact that EDHOC features two types of authentication, which can be combined arbitrarily. While signature-based authentication is compatible with past LIKE approaches, it is not clear that MAC-based authentication will provide provable non-framewability. Another important challenge is combining LIKE with the complex key-schedule of the EDHOC protocol.

Our contribution. We describe LI-compliant extensions for EDHOC's four authentication methods, and guarantee:

- **Key-secrecy:** session traffic keys are indistinguishable from random except for that session's Initiator, Responder, and the collusion of all the authorities allowed to perform LI for that session.
- **Non-Frameability:** it is impossible to falsely accuse a user of taking part in a session that it did not run.
- **LI Compliance:** the mobile infrastructure nodes (proxies) forwarding the communication can prove keys computed in accepting sessions are lawfully-interceptable by the correct set of LI authorities.
- **Identity-protection:** an attacker (active for Initiator and passive for Responder) cannot learn the identities of the two endpoints.

Each property provably holds in the LIKE models of [2,5], and a (modified) identity-protection model of [8]. Moreover, we provide an open-source implementation of our scheme in Rust optimized for constrained devices. We demonstrate the feasibility of our scheme by evaluating it on two hardware platforms that are typical examples of hardware used in the LAKE use cases.

Related work. Lawful Interception initially relied on Key-Escrow: the idea of entrusting communication to a managing trusted third party (TTP), which could learn the session key. Unfortunately, Key-Escrow often requires the online presence of authorities and can easily be pushed to mass surveillance. In spite of new LI techniques [3], current LI still relies on Key Escrow [26,16].

Our work comes closest to a different approach [2] called LIKE, which aimed to achieve fine-grained LI-compliant Authenticated Key-Exchange (AKE) with better privacy. A first pairing-based instantiation [2] was rendered more efficient, pairing-free, and usable in roaming scenarios by Bultel and Onete [5]. One property provided by LIKE is non-frameability, which is also underlined by recent related work [4] in the context of 5G (and beyond) network architectures.

to allow parties to be aware of potential LI (as we believe transparency should be reinforced) – but this could be technically adjusted to allow proxies to indicate, in a prior message, the authorities that must be used for each exchange.

Some techniques for connection-monitoring and data-encryption [13,15] could be extended to the LI scenario, though this is not their original goal. As they rely on pairings, they are incompatible with lightweight AKE, however. Finally, a worthwhile privacy-preserving alternative for LI is provided by CRUMPLE [27], which provides LI through a proof of work. This interesting approach is, unfortunately, not sufficiently efficient to comply with current LI requirements (which demand that interception yield “timely" results).

Finally note that throughout the standardization process of EDHOC [25], the protocol was analysed in various versions. A formal analysis using SAPIC+ identified weaknesses and proposed modifications in EDHOC version 12 [17]; later, the authors verified draft 14, showing that many previously identified issues had been addressed. Cottier and Pointcheval then provided an analysis of EDHOC draft 15 [8,7] and suggested further improvements. In 2023, Günther and Mukendi [12] analyzed draft 17 and proposed changes to the key schedule and to the construction of transcript hashes.

2 The LIKE framework and further primitives

Originally introduced in the context of mobile authenticated key-exchange protocols, Lawful Interception Key Exchange (LIKE) is a two-party AKE scheme featuring two proxies, representing the serving networks of the two endpoints [2]. The system also features a number of authorities, which need not be online for the duration of each protocol session. Each handshake is lawfully-interceptable by the collaboration of all the members of an authority set, whose size and composition depend on current legislations.

Formally, LIKE is defined as $\mathsf{LIKE} = (\mathtt{Setup}, \mathtt{UKeyGen}, \mathtt{OpKeyGen}, \mathtt{AKeyGen}, \mathsf{AKE}, \mathtt{Verify}, \mathtt{TDGen}, \mathtt{Open})$. $\mathtt{Setup}$ provides a global setup for the choice of universal parameters. The next three algorithms allow mobile users, operators, and authorities to generate long-term public keys. The LIKE protocol is run in sessions via the AKE algorithm. The verification algorithm allows for the verification of the soundness of the handshake and for the validation of the participants, whereas the last two algorithms provide mechanisms for LI.

LIKE assumes a scenario in which the two endpoints can only communicate through the proxies, and the protocol guarantees: key-security, non-frameability, and LI compliance (also called “honest-operator" in [2,5]). Key-security is ensured by essentially ensuring that only the legitimate owners of keys associated with authorities can retrieve meaningful trapdoor information for the session keys – and even then, only the composition of all the required trapdoors yields and the session key. Non-frameability requires, in LIKE, the use of signatures for both endpoints. The Honest-operator property is achieved by providing the proxy a means of verifying the LI-compliance of a protocol transcript, while not providing the proxy any information about the session key. In this paper, we start from [5], a follow-up work of [2] which achieved pairing-free LIKE in the roaming scenario, by using an Elgamal-like encryption of the session secret.

Our approach carefully combines this second LIKE approach, and makes use of both Non-Interactive Proofs of Knowledges (NIPoK) and Signatures of

ID	Initiator	Credential I	Responder	Credential R
0	Signature	$(\mathsf{ssk_I}, \mathsf{spk_I})$	Signature	$(\mathsf{ssk_R}, \mathsf{spk_R})$
1	Signature	$(\mathsf{ssk_I}, \mathsf{spk_I})$	Static Diffie-Hellman	(r, g^r)
2	Static Diffie-Hellman	(i, g^i)	Signature	$(\mathsf{ssk_R}, \mathsf{spk_R})$
3	Static Diffie-Hellman	(i, g^i)	Static Diffie-Hellman	(r, g^r)

Table 1: Current authentication methods registered by IANA (0-1-2-3).

Knowledge (SoK). Let $\mathcal{R}$ be a binary relation and let $\mathcal{L}$ be a language such that $s \in \mathcal{L} \iff \big(\exists w, (s, w) \in \mathcal{R}\big)$. We denote by $\nu := \mathsf{NIPoK}\{w : (w, s) \in \mathcal{R}\}$ the proof of knowledge of a witness w for a statement s, and by $\sigma := \mathsf{SoK}_m\{w : (w, s) \in \mathcal{R}\}$ the signature of knowledge on message m using witness w. Both NIPoKs and SoKs must be complete, extractable, and zero-knowledge.

3 The EDHOC protocol

EDHOC is a two-party mutually-authenticated Diffie-Hellman-based AKE scheme relying on Krawczyk's SIGMA-I protocol [19]. It features three mandatory messages and an optional fourth. The handshake is run in sessions between an Initiator I and a Responder R, which can choose how they want to authenticate: either by using signature schemes, or by using a certified static Diffie-Hellman (DH) key. The four combinations of authentication mechanisms are called methods 0, 1, 2, and 3, as described in Table 1. We denote by $(\mathsf{ssk_P}, \mathsf{spk_P})$ the private/public signature keys of party P, and by (p, g^p) party P's long-term static DH private/public keys. The parties indicate their preferred means of authentication method in `message_1`.

Protocol outline. Figure 1 depicts the EDHOC message flow. The first message, always sent unencrypted, consists of parameters `METHOD` and `SUITES_I`, indicating the authentication method and desired cipher suite, a *connection identifier* $\mathsf{C_I}$ which acts as a session identifier for peers running multiple sessions, some additional information called external authorization data $\mathsf{EAD_1}$, and a fresh ephemeral public key g^x (we denote the corresponding private key as x).

If the Responder supports the method and cipher suite, it computes `message_2`, which consists of an ephemeral DH element g^y sent in clear (the private key is denoted as y) and a ciphertext, encrypted with a key denoted as $\mathsf{sk_2}$, derived from g^x and g^y. The plaintext consists of: a connection identifier $\mathsf{C_R}$, an identifier indicating the Responder's long-term authentication credential, some additional data $\mathsf{EAD_2}$, and an authentication by the Responder, consisting of either a signature (methods 1 and 3) or a MAC (methods 0 and 2). The signature is computed with the Responder's private key $\mathsf{ssk_R}$ over the credential identifier $\mathsf{ID\_CRED\_R}$, a transcript hash $\mathsf{TH_2}$ detailed in Figure 2, the public key $\mathsf{spk_R}$, and $\mathsf{EAD_2}$. If the Responder uses static-DH authentication, then the MAC key is derived from both the ephemeral DH product of g^x and g^y and from the semi-static DH product of g^x and the static element g^r.

The Initiator decrypts the ciphertext in `message_2` with the key $\mathsf{sk_2}$, derived from g^x and g^y (the latter is sent in clear). Then, it recovers and verifies the

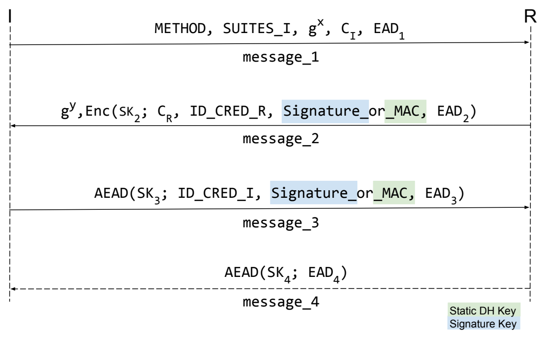
I
R
METHOD, SUITES_I, g^x, C_I, EAD_1
message_1
g^y,Enc(SK_2; C_R, ID_CRED_R, Signature_or_MAC, EAD_2)
message_2
AEAD(SK_3; ID_CRED_I, Signature_or_MAC, EAD_3)
message_3
AEAD(SK_4; EAD_4)
message_4
Static DH Key
Signature Key

Fig. 1: The EDHOC message flow [22].

Responder's authentication, according to the authentication method. `message_3` is somewhat similar to `message_2`, except that it is AEAD-encrypted using a key derived either solely from the ephemeral DH values g^x and g^y (methods 0 and 2) or from both the ephemeral DH product of g^x and g^y, and the semi-static DH product of g^x and g^r (methods 1 and 3).

The optional `message_4` provides explicit key-confirmation and is an AEAD-encryption on additional data EAD_4. The encryption key is derived from the previously computed ephemeral (and potentially semi-static) DH products, and additionally, for methods 2 or 3, a new semi-static DH product between the ephemeral value g^y and the Initiator's static DH element g^i.

The full key-schedule of the protocol is depicted in Figure 2. EDHOC uses two functions, `EDHOC_Expand` and `EDHOC_Extract`, with the EDHOC hash algorithm in the selected cipher suite to derive keys used in message processing. `EDHOC_Extract` is used to derive fixed-length uniformly pseudorandom keys (PRKs) from Elliptic Curve Diffie-Hellman (ECDH) shared secrets.
`EDHOC_Expand` is used to define a key derivation function, `EDHOC_KDF`, for generating MACs and for deriving output keying material (OKM) from PRKs.

There are three main secret values that are computed, labelled, respectively PRK_{2e}, PRK_{3e2m} and PRK_{4e3m}. The notations indicate the purposes of these secrets: the 2e subscript indicates the key is used to encrypt `message_2` content. The 3e2m subscript indicates the value is used to encrypt `message_3` and (depending on the authentication method) compute the MAC in `message_2`. Note that in case of signature-only authentication (method 0), the entire key-schedule is derived from the ephemeral DH values g^x and g^y. If both parties use static DH authentication, the key schedule resembles X3DH [8], using the ephemeral DH product and two semi-static ones.

EDHOC primitives. While EDHOC can be instantiated with different primitives, currently its implementations use elliptic-curve cryptography [23]. We will require the hardness of both the Decisional Diffie-Hellman (DDH) and the Gap Diffie-Hellman (GDH) problems in the groups selected in protocol ses-

[8] `https://signal.org/docs/specifications/x3dh/`

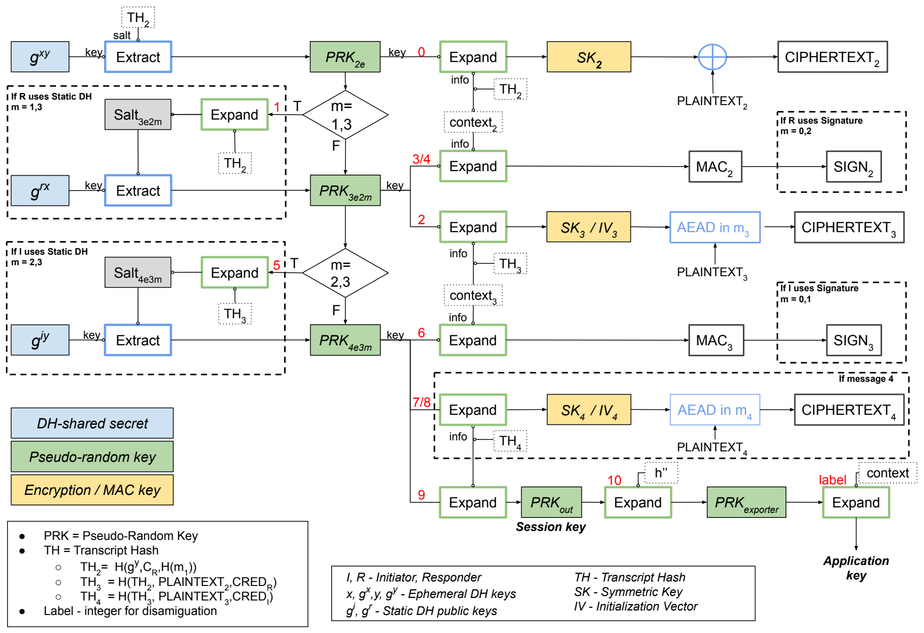

Fig. 2: EDHOC's key scheduling. The figure shows how keys are derived for all four authentication methods [23].

sions. Key-derivation for EDHOC employs HKDF [20]. In this paper, we idealise both extraction and expansion as a random oracle. Once expanded, keys are used in two fundamental ways: 1) for stream-cipher encryption (XOR-ing key-bits with plaintext-bits, which we require to be injective [8,21]), and 2) for AEAD-encryption (which we require to be AEAD-secure), both IND-CPA-secure. Key-derivation steps also rely on updated session hashes. We idealise the hash function $\mathcal{H}$ as a random oracle (in practice, instantiated as $\mathsf{SHA256}$). Finally, signature-based authentication requires the use of an EUF-CMA-secure signature scheme $\mathsf{DS} = (\mathsf{DS}.\mathtt{Gen}, \mathsf{DS}.\mathtt{Sign}, \mathsf{DS}.\mathtt{Verif})$ (usually instantiated as EdDSA or $\mathsf{ES256}$), while static-DH-based authentication is combined with a MAC scheme, implemented in EDHOC as `EDHOC_Expand`.

4 LI-compliant EDHOC

In order to empower Mobile Network Operators (MNOs) with the ability to provide privacy-preserving LI-compliant secure-channel establishment, we encrypt session secrets in the session state, such that only the collaboration of the entire authority-set will be able to decrypt these values. For that purpose, we essentially combine the LIKE approach and the EDHOC protocol (Section 3).

The composition is not black-box, for several important reasons. The first is our wish to preserve as many of the EDHOC properties as possible, including identity-protection. The latter requires that network adversaries (passive, for $\mathsf{Responders}$, active, for $\mathsf{Initiators}$) be unable to distinguish the endpoint's identity (Initiator or Responder). As a result, some of the LIKE messages will need to be incorporated in EDHOC's encrypted messages, modifying them in a non-trivial fashion. As a result, we prefer to use different notations for the two protocols, referring to EDHOC's first and second messages as `message_1` and `message_2` (as in Section 3), as opposed to M_1 and M_2 for our protocol LI-EDHOC. Another problem is verifying the consistency of the elements input in the key-computation, which we discuss below. Finally, we choose white-box composition for the sake of efficiency – since some LIKE elements are already present in EDHOC.

An important modification we make to the original LIKE [2,5] is that the session keys take in input an additional nonce n_r (and we use the exporter keys featured in EDHOC). This artifice of our construction is necessary in order to enforce LI-compliance on the Initiator's side. Thus, we include the nonce n_r in the context of the exporter PRK expansion (bottom right of Figure 2). Encrypting part of the handshake also raises a serious concern with respect to LI-compliance, in which proxies must be able to prove that the elements required to compute the key are consistent with the encrypted secrets and the ciphertexts. A naïve solution would be to send the entire messages unencrypted (thus nullifying the identity-protection afforded by EDHOC). Our solution tries to reconcile both LI requirements and identity-protection. We make endpoints send their encryption keys (not secrets) for handshake messages, encrypted with a key shared between the endpoint and the proxy[9], which allows the latter to verify the consistency of the transcript[10].

In order to provide LI-recovery of session keys, we essentially encrypt some component session secrets. This could be done naïvely by running the protocol of [5] for each secret in parallel; that, however, would be suboptimal and could potentially break the LI-compliance guarantee, since EDHOC features an interdependence between the ephemeral and semi-static secrets. Each secret will be recovered by the authorities from an ElGamal-like encryption of it under the product of the authorities' keys. During the protocol, each authority derives a trapdoor (essentially a local decryption with just its own private key) – and by using the homomorphic properties of ElGamal, the combination of all trapdoors yields the session's secret. Crucially, if even one authority withholds its trapdoor, the secret remains hidden – even from a collusion of all other authorities.

We proceed to describe our protocol, whose handshake is also depicted in Figures 3, 4 and 5. Due to space restrictions, we only detail LI-EDHOC for

[9] This is not a strong assumption, as the mobile protocol stack feature a hierarchy of keys usable to that effect.

[10] We considered two alternative approaches. The key under which the keys are encrypted could be negotiated during the handshake, at the expense of additional complexity. We discarded this alternative since a shared key is already present in the 5G stack. A second cleaner alternative would be to encrypt handshake messages with a different key, which is computable by the proxies and endpoints. Unfortunately this would require modifying the key schedule of EDHOC (already standardized) and introducing cumbersome key-computation tools including pairings.

$$\begin{tikzcd}
\text{Initiator} & \text{Proxy } \mathsf{P_I} & \text{Proxy } \mathsf{P_R} & \text{Responder} \\
\mathtt{InitRun1}(\mathsf{ID_I}) \arrow[r, "M_1"] & \mathtt{Prox_IRun1}(M_1) \arrow[r, "M_1"] & \mathtt{Prox_RRun1}(M_1) \arrow[r, "M_1"] & \mathtt{RespRun1}(\mathsf{ID_R}, M_1) \\
{} & \mathtt{Prox_IRun2}(M_2) \arrow[l, "M_2"'] & \mathtt{Prox_RRun2}(M_2) \arrow[l, "M_2"'] & {} \arrow[l, "M_2"'] \\
\mathtt{InitRun2}(\mathsf{ID_I}, M_2) \arrow[r, "M_3"] & \mathtt{Prox_IRun3}(M_3) \arrow[r, "M_3'"] & {} \arrow[r, "M_3'"] & \mathtt{RespRun2}(\mathsf{ID_R}, M_3') \\
\mathtt{InitRun3}(\mathsf{ID_I}, M_4'') & \mathtt{Prox_IRun4}(M_4') \arrow[l, "M_4''"'] & \mathtt{Prox_RRun3}(M_4) \arrow[l, "M_4'"'] & {} \arrow[l, "M_4"']
\end{tikzcd}$$

Fig. 3: Message flow of the LI-EDHOC handshake protocols. Algorithms for the **Static-Sign** ($\mathsf{ID} = 2$) instances are provided in Figures 4 and 5.

authentication method 2 (**Static-Sign**). The other 3 methods are included in our full version [21].

Setup and Key Generation. Endpoints (Initiators and Responders), proxies, and authorities will all generate long-term keys. Endpoints generate signature keypairs $(\mathsf{ssk_U}, \mathsf{spk_U})$ for authentication, using the algorithm $\mathsf{DS}.\mathtt{Gen}(1^\lambda)$ (for signature-based authentication) and DH pairs (p, g^p) with $p \in \{i, r\}$ (for static authentication). Proxies generate signature keys $(\mathsf{ssk_{Prox}}, \mathsf{spk_{Prox}})$ which authenticate LI material. Authorities generate DH keys for LI $\{\Lambda.\mathsf{SK}, \Lambda.\mathsf{pk} = g^{\Lambda.\mathsf{SK}}\}$, for which they prove knowledge of the private key: $\Lambda.\mathsf{ni} \leftarrow \mathsf{NIPoK}\{\Lambda.\mathsf{SK} : \Lambda.\mathsf{pk} = g^{\Lambda.\mathsf{SK}}\}$. We set $\Lambda.\mathsf{PK} \leftarrow (\Lambda.\mathsf{pk}, \Lambda.\mathsf{ni})$. Formal descriptions are provided in [21].

Precomputation. Handshakes are preceded by a precomputation phase, during which users and proxies verify all authorities' public keys and proofs. Note that due to roaming, LI can take place for an independent set of authorities at each end (Initiator and Responder). In addition, at this point we assume that, if the proxy does not already have a shared key pcsk, then we assume both the proxy and the endpoint generate this common key[11].

Let $(\Lambda^{\mathsf{I}}_j.\mathsf{PK})_{j=1}^{n_\mathsf{I}}$ denote the vector of the authority public keys involved in the interception (with Λ^{I}_j an authority for all $j \in [\![1, n_\mathsf{I}]\!]$, and with each $\Lambda^{\mathsf{I}}_j.\mathsf{PK} = (\Lambda^{\mathsf{I}}_j.\mathsf{pk}, \Lambda^{\mathsf{I}}_j.\mathsf{ni})$). The Initiator and its proxy $\mathsf{Prox_I}$ verify the proofs as $\mathsf{NIPoKver}(\Lambda^{\mathsf{I}}_j.\mathsf{pk}, \Lambda^{\mathsf{I}}_j.\mathsf{ni})$ and proceeds if all verifications pass. They set $\mathsf{h_I} \leftarrow \prod_{j=1}^{n_\mathsf{I}} \Lambda^{\mathsf{I}}_j.\mathsf{pk}$, and $\omega_\mathsf{I} \leftarrow (\Lambda^{\mathsf{I}}_j)_{j=1}^{n_\mathsf{I}} \| \mathsf{ID_I}$, returning $(\mathsf{h_I}, \omega_\mathsf{I})$. This is analogous on the Responder side and its associated proxy $\mathsf{Prox_R}$.

The values $(\mathsf{h_I}, \omega_\mathsf{I})$ and $\mathsf{pcsk}_{\mathsf{I},\mathsf{Prox_I}}$ (for the Initiator), and $(\mathsf{h_R}, \omega_\mathsf{R})$ and $\mathsf{pcsk}_{\mathsf{R},\mathsf{Prox_R}}$ (for the Responder), are shared between the proxies and the endpoints, and both entities check (h, ω) to ensure that neither of them is cheating. We assume moreover that the proxies know the identity of the endpoints (as is already the case following an initial AKA/Handshake). This stage can be done once and then reused for multiple sessions (for the same set of authorities of an endpoint).

[11] This would be done at the beginning of each handshake anyway, within the 5G protocol stack.

```
$\texttt{InitRun1}(\mathsf{ID_I})$
1 : $(x \leftarrow\!\!\$\ \mathbb{Z}_p^*, g^x), \mathsf{C_I} \leftarrow\!\!\$\ \{0,1\}^{l_\mathsf{C}}$
2 : $\textbf{return}\ M_1 \leftarrow (\mathsf{MCS} \| g^x \| \mathsf{C_I} \| \mathsf{EAD_1})$

$\texttt{InitRun2}(\mathsf{ID_I}, M_2)$
1 : Parse $M_2$ as $(g^y \| c_2), g^{xy} \leftarrow (g^y)^x, g^{yi} \leftarrow (g^y)^i$
    // Compute $\mathsf{sk_2}$, decrypt $c_2$
2 : Compute $\mathsf{TH_2}, \mathsf{PRK_{2e}}, \mathsf{sk_2}$ as in $\texttt{RespRun1}$, lines 4,5,9
3 : $m_2 \leftarrow \mathsf{Dec}(\mathsf{sk_2}, c_2)$, parse $m_2$ as $(\mathsf{C_R} \| \mathsf{ID_R} \| \sigma_2 \| \mathsf{EAD_2})$
4 : $\mathsf{sid} \leftarrow (\mathsf{C_I}, \mathsf{C_R}, g^x, g^y)$
5 : Compute $\mathsf{CTX_2}, t_2'$ as in $\texttt{RespRun1}$, lines 6,7
    // Verify Responder authentication
6 : $\boxed{\textbf{if}\ \mathsf{DS}.\texttt{Verif}(\mathsf{spk_R}, (l_\mathsf{sig} \| \mathsf{CTX_2} \| t_2'), \sigma_2) \neq 1}$
7 :         $\textbf{then return}\ \perp$
    // Compute key-schedule for 3rd message
8 : $\mathsf{TH_3} \leftarrow \mathcal{H}(\mathsf{TH_2}, m_2, \mathsf{ID_R}), \mathsf{CTX_3} \leftarrow (\mathsf{ID_I} \| \mathsf{TH_3} \| g^i \| \mathsf{EAD_3})$
9 : $\mathsf{sk_3} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{2e}}, 3 \| \mathsf{TH_3} \| l_\mathsf{key}, l_\mathsf{key})$
10 : $\mathsf{IV_3} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{2e}}, 4 \| \mathsf{TH_3} \| l_\mathsf{iv}, l_\mathsf{iv})$
11 : $\mathsf{ad_3} \leftarrow (l_\mathsf{aead} \| \text{“ ”} \| \mathsf{TH_3})$
12 : $\mathsf{salt_{4e3m}} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{2e}}, 5 \| \mathsf{TH_3} \| l_\mathsf{hash}, l_\mathsf{hash})$
13 : $\mathsf{PRK_{4e3m}} \leftarrow \texttt{HKDF\_Extract}(\mathsf{salt_{4e3m}}, g^{yi})$
    // Authenticate (static DH in method 2)
14 : $\boxed{t_3 \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 6 \| \mathsf{CTX_3} \| l_\mathsf{mac}, l_\mathsf{mac})}$
    // Ensure LI-compliance and assemble $M_3$
15 : $m_3 \leftarrow (\mathsf{ID_I} \| t_3 \| \mathsf{EAD_3}), c_3 \leftarrow \mathsf{Enc}'(\mathsf{sk_3}, \mathsf{IV_3}, \mathsf{ad_3}, m_3)$
16 : $\mathsf{TH_4} \leftarrow \mathcal{H}(\mathsf{TH_3}, m_3, \mathsf{ID_I})$
17 : $\dbox{\mathsf{H_I} \leftarrow (\mathsf{h_I} g^y)^x, \mathsf{H_I^1} \leftarrow (\mathsf{h_I} g^y)^i}$
18 : $\dbox{\mathsf{stm_I} \leftarrow \big[X = g^x \wedge \mathsf{H_I} = (\mathsf{h_I} g^y)^x \wedge X_1 = g^i \wedge \mathsf{H_I^1} = (\mathsf{h_I} g^y)^i\big]}$
19 : $\dbox{\mathsf{ni_I} \leftarrow \mathsf{SoK}_{\omega_\mathsf{I} \| \mathsf{ID_R} \| \mathsf{TH_2} \| \mathsf{TH_3} \| \mathsf{TH_4}}\{(x,i) : \mathsf{stm_I}\}}$
20 : $\dbox{K_\mathsf{I} \leftarrow (\mathsf{sk_2} \| \mathsf{sk_3} \| \mathsf{IV_3} \| t_2), S_\mathsf{I} \leftarrow \mathsf{Enc}''(\mathsf{pcsk_{I,Prox_I}}, K_\mathsf{I})}$
21 : $\textbf{return}\ M_3 \leftarrow (c_3 \| \mathsf{H_I} \| \mathsf{H_I^1} \| \mathsf{ni_I} \| S_\mathsf{I})$

$\texttt{InitRun3}(\mathsf{ID_I}, M_4'')$
1 : Parse $M_4''$ as $(c_4 \| n_r \| \mathsf{val} \| \mathsf{TH_5})$, $\textbf{if}\ c_4 = \perp\ \textbf{then return}\ \perp$
2 : $\textbf{if}\ \mathsf{TH_5} \neq \mathcal{H}(\mathsf{TH_4}, \mathsf{val}, n_r)\ \textbf{then return}\ \perp$
3 : Compute $\mathsf{sk_4}, \mathsf{IV_4}, \mathsf{ad_4}$ as in $\texttt{RespRun2}$, lines 7-9
4 : $m_4 \leftarrow \mathsf{Dec}'(\mathsf{sk_4}, \mathsf{IV_4}, \mathsf{ad_4}, c_4)$, parse $m_4$ as $\mathsf{EAD_4}$
5 : $\textbf{if}\ m_4 = \perp\ \textbf{then return}\ \perp$
6 : $\mathsf{PRK_{out}} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 7 \| \mathsf{TH_4} \| \dbox{n_r} \| l_\mathsf{hash}, l_\mathsf{hash})$
7 : $\texttt{terminated} \leftarrow 1$
```

```
$\texttt{RespRun1}(\mathsf{ID_R}, M_1)$
1 : $(y \leftarrow\!\!\$\ \mathbb{Z}_p^*, g^y), \mathsf{C_R} \leftarrow\!\!\$\ \{0,1\}^{l_\mathsf{C}}$
2 : Parse $M_1$ as $(\mathsf{MCS} \| g^x \| \mathsf{C_I} \| \mathsf{EAD_1})$
    // Compute key-schedule for 2nd message
3 : $g^{xy} \leftarrow (g^x)^y, \mathsf{sid} \leftarrow (\mathsf{C_I}, \mathsf{C_R}, g^x, g^y)$
4 : $\mathsf{TH_2} \leftarrow \mathcal{H}(g^y, \mathcal{H}(M_1))$
5 : $\mathsf{PRK_{2e}} \leftarrow \texttt{HKDF\_Extract}(\mathsf{TH_2}, g^{xy})$
6 : $\mathsf{sk_2} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{2e}}, 0 \| \mathsf{TH_2} \| l_2, l_2)$
7 : $\mathsf{CTX_2} \leftarrow (\mathsf{C_R} \| \mathsf{ID_R} \| \mathsf{TH_2} \| \mathsf{EAD_2})$
8 : $t_2 \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{2e}}, 2 \| \mathsf{CTX_2} \| l_\mathsf{mac}, l_\mathsf{mac})$
    // Authenticate (signature in method 2)
9 : $\boxed{\sigma_2 \leftarrow \mathsf{DS}.\texttt{Sign}(\mathsf{ssk_R}, (l_\mathsf{sig} \| \mathsf{CTX_2} \| t_2))}$
    // Assemble $M_2$
10 : $m_2 \leftarrow (\mathsf{C_R} \| \mathsf{ID_R} \| \sigma_2 \| \mathsf{EAD_2}), c_2 \leftarrow \mathsf{Enc}(\mathsf{sk_2}, m_2)$
11 : $\textbf{return}\ M_2 \leftarrow (g^y \| c_2)$

$\texttt{RespRun2}(\mathsf{ID_R}, M_3')$
1 : Parse $M_3'$ as $c_3$
    // Compute $\mathsf{sk_3}$ and decrypt $c_3$
2 : Compute $\mathsf{TH_3}, \mathsf{sk_3}, \mathsf{IV_3}, \mathsf{ad_3}$ as in $\texttt{InitRun2}$, lines 8-11
3 : $m_3 \leftarrow \mathsf{Dec}'(\mathsf{sk_3}, \mathsf{IV_3}, \mathsf{ad_3}, c_3)$, parse $m_3$ as $(\mathsf{ID_I} \| t_3 \| \mathsf{EAD_3})$
4 : Get $g^i$ from $\mathsf{ID_I}$, $g^{yi} \leftarrow (g^i)^y$
    // Compute key-schedule for 4th message
5 : Compute $\mathsf{CTX_3}, \mathsf{salt_{4e3m}}, \mathsf{PRK_{4e3m}}$ as in $\texttt{InitRun2}$, lines 8,12,13
6 : $\mathsf{TH_4} \leftarrow \mathcal{H}(\mathsf{TH_3}, m_3, \mathsf{ID_I}), \mathsf{ad_4} \leftarrow (l_\mathsf{aead}, \| \text{“ ”} \| \mathsf{TH_4})$
7 : $\mathsf{sk_4} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 8 \| \mathsf{TH_4} \| l_\mathsf{key}, l_\mathsf{key})$
8 : $\mathsf{IV_4} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 9 \| \mathsf{TH_4} \| l_\mathsf{key}, l_\mathsf{key})$
    // Verify Initiator authentication
9 : $t_3' \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 6 \| \mathsf{CTX_3} \| l_\mathsf{mac}, l_\mathsf{mac})$
10 : $\boxed{\textbf{if}\ t_3 \neq t_3'\ \textbf{then return}\ \perp}$
    // Ensure LI-compliance and assemble $M_4$
11 : $\dbox{\mathsf{H_R} \leftarrow (\mathsf{h_R} g^x)^y, \mathsf{H_R^1} \leftarrow (\mathsf{h_R} g^i)^y}$
12 : $\dbox{\mathsf{stm_R} \leftarrow \big[Y = g^y \wedge \mathsf{H_R} = (\mathsf{h_R} g^x)^y \wedge \mathsf{H_R^1} = (\mathsf{h_R} g^i)^y\big]}$
13 : $\dbox{\mathsf{ni_R} \leftarrow \mathsf{SoK}_{\omega_\mathsf{R} \| \mathsf{ID_I} \| \mathsf{TH_2} \| \mathsf{TH_3} \| \mathsf{TH_4}}\{y : \mathsf{stm_R}\}}$
14 : $\dbox{K_\mathsf{R} \leftarrow (\mathsf{sk_2} \| \mathsf{sk_3} \| \mathsf{IV_3} \| t_2), S_\mathsf{R} \leftarrow \mathsf{Enc}''(\mathsf{pcsk_{R,Prox_R}}, K_\mathsf{R})}$
15 : $m_4 \leftarrow \mathsf{EAD_4}, c_4 \leftarrow \mathsf{Enc}'(\mathsf{sk_4}, \mathsf{IV_4}, \mathsf{ad_4}, m_4)$
16 : $\dbox{n_r \leftarrow\!\!\$\ \{0,1\}^{l_\mathsf{rand}}}, \mathsf{val} \leftarrow \mathcal{H}(g^{xy}, \text{“ ”}, n_r)$
17 : $\mathsf{PRK_{out}} \leftarrow \texttt{HKDF\_Expand}(\mathsf{PRK_{4e3m}}, 7 \| \mathsf{TH_4} \| \dbox{n_r} \| l_\mathsf{hash}, l_\mathsf{hash})$
18 : $\texttt{terminated} \leftarrow 1, \mathsf{TH_5} \leftarrow \mathcal{H}(\mathsf{TH_4}, \mathsf{val}, n_r)$
19 : $\textbf{return}\ M_4 \leftarrow (c_4 \| n_r \| \mathsf{val} \| \mathsf{TH_5} \| \mathsf{H_R} \| \mathsf{H_R^1} \| \mathsf{ni_R} \| S_\mathsf{R})$
```

Fig. 4: The LI-EDHOC protocol for authentication method 2.
Dashed boxes represent operations associated with LI. The notation MCS denotes the method and the cipher suite that the Initiator wants to use.

Handshake. We assume all precomputations are done. The handshake is run as described in Figure 3, with details provided in Figures 4 (endpoint computations) and 5 (for the proxies, in Appendix C).

The first message sent by the Initiator is unchanged compared to EDHOC and contains the Initiator's ephemeral secret g^x. This message is faithfully forwarded by both proxies, which store g^x and ω_I (for the Initiator's proxy), respectively ω_R (on the Responder side). The Responder follows EDHOC and generates the ephemeral secret g^y, then follows the key-schedule in order to obtain the secret PRK_{2e} and the derived key sk_2. In method 2, the Responder computes a signature using its long-term signing key[12]. The computed message is the same as in EDHOC (line 10 of the `RespRun1` routine in Figure 4), and it is XOR-encrypted with sk_2. The Responder proxy stores g^y and the message. On the Initiator side, the proxy forwards the message faithfully, but stores g^y.

LI elements start being added from the third message onward. In method 2, keys are computed based on two DH products g^{xy} and g^{iy} (the Initiator contributes its static long-term DH key g^i as well as the ephemeral g^x). The Initiator computes trapdoor messages H_I and H_I^1 (line 17 of `InitRun2` in Figure 4), which essentially encrypt g^{xy} and g^{iy} under the product h_I of the authorities' public keys. In order to prevent the Initiator from cheating, the latter has to compute a signature of knowledge proving that the exponents of the ephemeral g^x, the static g^i and both H_I and H_I^1 are the same (line 18). The message that is signed consists of the set of identities ω_I, as well as the Responder's identity and the concatenation of the transcript hashes used to compute the keys. Finally, in order to prevent cheating, the Initiator encrypts the tag t_2 and the keys used to encrypt/AE-encrypt messages m_2, m_3 with the key $\mathsf{pcsk}_{\mathsf{I},\mathsf{Prox}_\mathsf{I}}$ it shares with its proxy as S_I. Thus M_3 is composed of ciphertexts c_3 and S_I, and LI elements H_I, H_I^1 and ni_I. The ciphertext S_I allows the proxy to decrypt/AE-decrypt the received second- and third-message ciphertexts, and verify the correctness of both transcript-hash values and the signature σ_2 (aborting if the endpoint misbehaves). Then it will store the transcript hashes, long-term static Initiator DH key, the two trapdoors, the signature of knowledge, the Responder identifier, and the HMAC, forwarding only the EDHOC ciphertext c_3 to the Responder's proxy.

The procedure is analogous for the preparation of the fourth message on the Responder's side after receiving and analyzing the third message as in the original protocol. Note that this message is compulsory in LI-EDHOC, as opposed to the original protocol. The final ingredient for Lawful Interception is an added nonce n_r generated by the Responder and input in the key-generation step in `RespRun2` lines 19-20 in Figure 4.

The reason we must add n_r in the key-computation step is as follows: unlike in [2,5], the Responder's signature of knowledge is verified only after the latter computed the final key. Thus, if the Initiator computed the key before the Responder, the Initiator's proxy might not be able to provide LI-interceptable

[12] This correspond to the authentication method provided in the original EDHOC protocol. In method 2, the Responder authenticates by signing t_2, and the Initiator authenticates thanks to t_3 partially computed with its static DH share.

communication. The nonce n_r is sent unencrypted in the fourth mandatory message to the $\mathsf{Initiator}$, which can now compute the same session key. In order to ensure that this nonce also arrives securely at the proxies and the $\mathsf{Initiator}$, the $\mathsf{Responder}$ sends the nonce n_r, a verification value val and $\mathsf{TH}_5 = \mathcal{H}(\mathsf{TH}_4, \mathsf{val}, n_r)$, so that every party may check the validity of the nonce (using TH_4 they all computed themselves). In addition, the proxy Prox_R also verifies the signature σ_2 with the tag t_2. Finally each proxy stores in the session state the nonce n_r, the tag t_2 and the value val, for a potential verification from the authorities.

Verification of sst. The session state sst consists of: $\omega \| \mathsf{MCS} \| g^x \| g^y \| g^i \| \mathsf{TH}_2 \| \mathsf{TH}_3 \| \mathsf{TH}_4 \| M_1 \| m_2 \| m_3 \| \mathsf{H} \| \mathsf{H}^1 \| \mathsf{ni} \| \mathsf{pid} \| t_2 \| n_r \| \mathsf{val} \| \sigma$. The plaintext messages m_2, m_3 allow for the verification of the transcript hashes and the authentication of both parties (through signatures of knowledge). The verification algorithm also extracts, from ω the public keys of the authorities, for which the proofs of knowledge of the discrete logarithm must be verified. Subsequently the signature of knowledge is verified with respect to both the signed message and to the equality of the discrete logarithm used to compute H and H^1. Since method **Static-Sign** only requires the $\mathsf{Responder}$ to use signature-based authentication, the verification algorithm obtains, from m_2 and t_2 all the necessary information to verify the signatures σ_2. Finally, parsing sst as $\tau \| \sigma$, the signature σ generated by the proxy is verified with respect to the message τ.

Lawful Interception. Given an sst, authorities invariably must first verify it (using the process described above). If sst is correct, each authority Λ^d first derives individual trapdoors for each of the two secrets: compute $\mathsf{A}_\mathsf{I} \leftarrow g^x$, $\mathsf{A}_\mathsf{R} \leftarrow g^y$, $\Lambda^\mathsf{d}.\mathsf{td}_1 \leftarrow {\mathsf{A}_\mathsf{d}}^{\Lambda^\mathsf{d}.\mathsf{SK}}$, $\Lambda^\mathsf{d}.\mathsf{td}_2 \leftarrow g^{i\Lambda^\mathsf{d}.\mathsf{SK}}$ ($\mathsf{d} \in \{\mathsf{I}, \mathsf{R}\}$) and prove

$$\Lambda^\mathsf{d}.\mathsf{td}_0 \leftarrow \mathsf{NIPoK}\{\Lambda^\mathsf{d}.\mathsf{SK} : \Lambda^\mathsf{d}.\mathsf{pk} = g^{\Lambda^\mathsf{d}.\mathsf{SK}} \wedge \Lambda^\mathsf{d}.\mathsf{td}_1 = {\mathsf{A}_\mathsf{d}}^{\Lambda^\mathsf{d}.\mathsf{SK}} \wedge \Lambda^\mathsf{d}.\mathsf{td}_2 = g^{i\Lambda^\mathsf{d}.\mathsf{SK}}\}.$$

Finally, set and return $\Lambda^\mathsf{d}.\mathsf{td} \leftarrow (\Lambda^\mathsf{d}.\mathsf{td}_0, \Lambda^\mathsf{d}.\mathsf{td}_1, \Lambda^\mathsf{d}.\mathsf{td}_2)$. Once all trapdoors are available, all parties verify the proofs $\Lambda^\mathsf{d}.\mathsf{td}_0$ and reconstitute the secrets g^{xy} and g^{iy}. The calculations, *e.g.*, on the $\mathsf{Initiator}$'s side for g^{xy}, are as follows:

$$\frac{\mathsf{H}_\mathsf{I}}{\prod_{j=1}^{n_\mathsf{I}} \Lambda^\mathsf{I}_j.\mathsf{td}_1} = \frac{(\mathsf{h}_\mathsf{I} g^y)^x}{\prod_{j=1}^{n_\mathsf{I}} g^{x\Lambda^\mathsf{I}_j.\mathsf{SK}}} = \frac{\left(\prod_{j=1}^{n_\mathsf{I}} g^{\Lambda^\mathsf{I}_j.\mathsf{SK}}\right)^x g^{xy}}{\prod_{j=1}^{n_\mathsf{I}} g^{x\Lambda^\mathsf{I}_j.\mathsf{SK}}} = \frac{\prod_{j=1}^{n_\mathsf{I}} g^{x\Lambda^\mathsf{I}_j.\mathsf{SK}} g^{xy}}{\prod_{j=1}^{n_\mathsf{I}} g^{x\Lambda^\mathsf{I}_j.\mathsf{SK}}} = g^{xy}.$$

To retrieve g^{iy}, we proceed in the same way, using H^1, and either $\Lambda^\mathsf{d}.\mathsf{td}_1$ or $\Lambda^\mathsf{d}.\mathsf{td}_2$ depending on the point of view ($\mathsf{Initiator}$ or $\mathsf{Responder}$).

At this point, authorities check the validity of the last unverified element of the sst: n_r. To do this they simply compute $\mathcal{H}(g^{xy}, \text{“ ”}, n_r)$ and compare this value to the provided val value in sst. This verification ensures that this n_r, transmitted by the proxy, is indeed the one used by the endpoints to compute their session key $\mathsf{PRK}_\mathsf{out}$. Finally, with the two secrets g^{xy} and g^{iy}, the authorities, using the verified additional data TH_2, TH_3, TH_4 and n_r, are now able to recompute the original EDHOC's key derivation schedule to derive the original session key $\mathsf{PRK}_\mathsf{out}$ (which coincides with the one computed by the endpoints).

5 Security Analysis in the Random Oracle Model

Our LI-EDHOC protocol guarantees the key-security, non-frameability, and LI-compliance security properties from LIKE [2,5], but also a stronger property of EDHOC: identity protection. We provide here only a high-level description of the security model. A complete formalization is in the full version [21].

Execution environment. LIKE includes three types of participants: endpoints belonging to a set ESet, proxies from a set $\mathsf{PROXSet}$, and authorities in a set $\mathsf{AUTHSet}$. Each party has private and public parameters $(\mathsf{P}.\overline{\mathsf{SK}}, \mathsf{P}.\overline{\mathsf{PK}})$ – which for EDHOC are in fact private/public *keysets* instead of just *one key*. We also make endpoints and proxies store a set of symmetric keys shared between them in an attribute $\mathsf{P}.\mathtt{PCSK} = \{(\mathsf{Q}, \mathsf{pcsk}_{\mathsf{P},\mathsf{Q}})\}, \mathsf{Q} \in \mathsf{ShareSet}$ (if $\mathsf{P} \in \mathsf{ESet}$, then $\mathsf{ShareSet} = \mathsf{PROXSet}$ and vice-versa). Any party may be corrupted, yielding to the adversary one element of the private keyset. The party stores a corruption bit $\mathsf{P}.\mathsf{SK}.\gamma$ for each key $\mathsf{SK} \in \mathsf{P}.\overline{\mathsf{SK}}$.

Protocol sessions are always run by *instances* of four parties: two endpoints, playing the parts of $\mathsf{Initiator}$ and $\mathsf{Responder}$; and two proxies. We denote by π_{P}^{q} the q^{th} instance of P. Each endpoint/proxy instance keeps track of *attributes*, including: the session identifier $\pi_{\mathsf{P}}^{q}.\mathsf{sid}$; the computed session key $\pi_{\mathsf{P}}^{q}.\mathsf{PRK}_{\mathsf{out}}$; a session state $\pi_{\mathsf{P}}^{q}.\mathsf{sst}$; partner identifiers AID for the authorities chosen by that party as legitimate for Lawful Interception, PNID for the partnering endpoint (or endpoints if this is a proxy instance), and ProxID for the partnering proxies; a flag indicating authentication acceptance (denoted $\pi_{\mathsf{P}}^{q}.\alpha$); and $\pi_{\mathsf{P}}^{q}.\mathsf{role}$ (the role $\mathsf{role} \in \{\mathsf{Initiator}, \mathsf{Responder}, \mathsf{Proxy}_{\mathsf{I}}, \mathsf{Proxy}_{\mathsf{R}}\}$ of P in the handshake).

To better capture EDHOC, endpoint party instances require several additional values: $\pi_{\mathsf{P}}^{q}.\mathsf{meth}$ (the method that will be used for a session involving this instance); $\pi_{\mathsf{P}}^{q}.\mathsf{SK}$ (the private key used by P to authenticate); $\pi_{\mathsf{P}}^{q}.\mathsf{pid}.\mathsf{PK}$ (the public key that P expects its partner to use for authentication), a flag indicating the corruption of the long-term private key used by that instance (denoted $\pi_{\mathsf{P}}^{q}.\mathsf{SK}.\gamma$); a flag indicating the revelation of the session key (denoted $\pi_{\mathsf{P}}^{q}.\rho$); and $\pi_{P}^{q}.\mathsf{ProxID}$ (the identity $\mathsf{ProxID} \in \mathsf{PROXSet}$ of the party's proxy). Both proxy and endpoint instances also store the current symmetric encryption key $\pi_{\mathsf{P}}^{q}.\mathsf{pcsk} \in \mathsf{P}.\mathtt{PCSK}$ used during the handshake. Finally proxy instances store two specific values : $\pi_{\mathsf{P}}^{q}.\mathsf{sst}$ (the session state built during the protocol session run) and $\pi_{\mathsf{P}}^{q}.\mathsf{tr}$ (the transcript of the protocol session run).

Following [2,5] we define the notion of instances *have matching conversation.*

Definition 1. *Let* (P, Q) *be two endpoints with* $\mathsf{P} \neq \mathsf{Q}$ *and let* q, j *be two natural integers. We says that* π_{P}^{q} *have matching conversation with* π_{Q}^{j} *(or that* π_{P}^{q} *matches* π_{Q}^{j}*) if* $\pi_{\mathsf{P}}^{q}.\mathsf{sid} \neq \perp$, $\pi_{\mathsf{P}}^{q}.\mathsf{sid} = \pi_{\mathsf{Q}}^{j}.\mathsf{sid}$, $\mathsf{Q} = \pi_{\mathsf{P}}^{q}.\mathsf{PNID}$ *and* $\mathsf{P} = \pi_{\mathsf{Q}}^{j}.\mathsf{PNID}$.

For *correctness*, accepting endpoint instances that have matching conversation compute the same session key; accepting endpoint instance keys are lawfully-interceptable by the collaboration of all the authorities in the instance's AID.

Oracles. Similarly to [2,5], we use game-based security notions and give the adversary access to oracles, such as: $\mathtt{oRegister}(\mathsf{P}, \mathsf{type}, \mathsf{P}.\overline{\mathsf{PK}})$ ($\mathcal{A}$ can register either honest parties, or malicious parties, with malicious $\mathsf{P}.\overline{\mathsf{PK}}$); $\texttt{oSend}(\pi_{\mathsf{P}}^{q}, m)$ ($\mathcal{A}$ can send a message to an endpoint instance $\pi_{\mathsf{P'}}^{j}$); $\texttt{oReveal}(\pi_{\mathsf{P}}^{q})$ ($\mathcal{A}$ learns the session key of endpoint instance π_{P}^{q}); $\texttt{oTest}_b(\pi_{\mathsf{P}}^{q})$ – which can only be queried once, and returns either the real session key computed by endpoint instance π_{P}^{q} or a random key, depending on the value of a bit b; and $\texttt{oRevealTD}(\mathsf{sst}, \mathsf{I}, \mathsf{R}, \mathsf{Prox}, \mathsf{AID}, \ell)$ ($\mathcal{A}$ learns the trapdoor of authority Λ_ℓ registered in authority set AID for a session with state sst. We need to modify two oracles in our work, extending them to capture multiple authentication modes and credentials:

- $\texttt{oNewSession}(\mathsf{P}, \mathsf{role}, \mathsf{PNID}, \mathsf{PK_I}, \mathsf{PK_R}, \mathsf{ProxID}, \mathsf{AID})$ – which creates endpoint/proxy instances of an endpoint/proxy P with role $\mathsf{role} = \mathsf{Initiator}$ or $\mathsf{role} = \mathsf{Responder}$, or $\mathsf{role} = \mathsf{Proxy}$, with $\mathsf{Initiator}$ authentication using the $\mathsf{Initiator}$'s public credential $\mathsf{PK_I}$ and $\mathsf{Responder}$ authentication using $\mathsf{Responder}$ credential $\mathsf{PK_R}$. This allow to create an instance using a specific method (either 0, 1, 2 or 3);
- $\texttt{oCorrupt}(\mathsf{P}, \ell)$ – which allows the adversary to corrupt the ℓ^{th} long-term key of party P.

Key security. In the key-security game, the adversary uses all the oracles presented above in order to learn the value of the bit b used for the single $\texttt{oTest}$ query. The adversary wins if, and only if, it guarantees the soundness property (*i.e.,* two honest instances having matching conversation and running a protocol session derive the same session key), it guesses b correctly and if its $\texttt{oTest}$ query target *fresh* instances (*i.e.,* for which the long-term authentication credential used by both endpoints in that instance is uncorrupted at the time of the test query, the session key has not been revealed, and at least one authority in AID is honest and its trapdoor has not been revealed). This is fully detailed in [21].

Non-frameability. In this game, the adversary may query all but the $\texttt{oTest}$ oracle, and wins if it ties an honest endpoint P to a session with state sst, in which P did not end in an accepting state.

LI-compliance. In LI-compliance, the adversary may query all but the $\texttt{oTest}$ oracle, and must output a proxy session in which the key retrieved by LI differs from the key that can be extracted from the session transcript.

The security of LI-EDHOC. We state the security theorems of LI-EDHOC for all the four authentication methods and prove them in the full version [21]. Interestingly, allowing both static-DH-based and signature-based authentication yields both a positive and a negative outcome. On the plus side, if one credential is lost or corrupted, the endpoint need not immediately re-initialize its public keys. Moreover, note that EDHOC's static-DH authentication mode is still forward-secure, since it is combined with an ephemeral secret. On the downside, however, the use of static-DH authentication renders non-frameability difficult to prove, requiring either heavier computations (which we present in the full version, for completeness), or a slight modification of the standard Discrete Logarithm problem (we will require access to a CDH oracle, akin to how GDH requires DDH oracle access). We dub this new problem *Oracle-DLog*[13].

[13] This problem is similar to the *Q-One-More Diffie–Hellman* problem defined in [18].

Definition 2 (Oracle-DLog). *Let $\mathbb{G}$ be a cyclic group of prime order p generated by $g \in \mathbb{G}$. Let $x \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and $X \leftarrow g^x$. The* Oracle-DLog *problem challenges an adversary in possession of $(g, p, \mathbb{G}, X)$, with oracle access to $\mathcal{O}_{\mathsf{CDH}}(\cdot)$ – returning, on input $Y = g^y \in \mathbb{G}$, the value $g^{xy} \in \mathbb{G}$ – to retrieve x.*

Theorem 1. *If* LI-EDHOC *is instantiated with an* EUF-CMA*-secure signature scheme; zero-knowledge and extractable proofs and signatures of knowledge; IND-CPA and injective encryption schemes; and if* DDH *and* GDH *problems are hard in $\mathbb{G}$, then* LI-EDHOC *(methods 0,1,2,3) achieve* key security, LI-compliance*, and* ($\mathsf{Initiator}$ and $\mathsf{Responder}$) identity protection*. Moreover,* LI-EDHOC *method 0 also achieves* non-frameability*. If, in addition, the* Oracle DLog *problem is hard in $\mathbb{G}$,* LI-EDHOC *(methods 1,2,3) also guarantee* non-frameability*.*

6 Implementation and Evaluation

As method 3 (**static-static**) of authentication is likely to be the most expensive, we implement LI-EDHOC in this particular setting (which appears in the full version). We complement the `lakers` implementation of EDHOC in Rust, with bindings for C and Python. As an online addition to this article, we release the LIKE implementation as open source[14].

Our implementation transports LIKE elements within the External Authorization Data (EAD) fields of EDHOC. This allows for selective activation of lawful interception maintaining EDHOC's base structure. There are two main additions to the lakers implementation of EDHOC: (1) the cryptographic primitives supporting the generation and verification of the Signature of Knowledge. We instantiate the SoK as described in Appendix A. (2) dedicated EAD handlers that generate and process LIKE elements during the EDHOC handshake. These handlers populate the EAD field of the message with the necessary additional elements of LIKE. The implementation includes the pre-computation phase performed once for a given set of peers and authorities, during which the values $(\mathsf{h_I}, \omega_\mathsf{I})$ are computed.

We evaluate the implementation on two typical constrained platforms, based on the ARM Cortex-M core[15]: (1) nRF52840 development board featuring an ARM Cortex-M4F processor running at 64 MHz with 256 kB RAM and 1 MB flash memory, and (2) STM32WBA55CG development board featuring an ARM Cortex-M33 processor running at 32 MHz with 128 kB RAM and 512 kB flash memory. We measure both overhead and execution time. To measure the later, we used the Saleae Logic 8 logic analyzer connected to specific GPIO pins on both development boards, that are toggled at the beginning and end of given code sections (Figure 6, Appendix C).The sampling frequency is set to 40 MS/s (Mega Samples per second). For the evaluation, the $\mathsf{Initiator}$ and the $\mathsf{Responder}$ are located on the same physical device, which removes the network communication overhead and allows us to focus on computational performance. Note that the evaluation does not include the proxies which run on non-constrained devices.

[14] `https://github.com/ElsaLopez133/lakers.git`

[15] Note that we are not making a comparison here, but simply presenting implementation results on commonly used platforms.

We made a comparison of the message overhead for the Initiator (messages 1 and 3) running EDHOC authenticated with DH Static Keys, with and without implementing LIKE. With LIKE, a total of 152 bytes (37 for the first message, and 115 for the third) are carried out for the Initiator, while without, only 56 bytes (37 for the first message, and 19 for the third) are carried out. The extra bytes compared to the standard EDHOC implementation in the third message come from the additional signature of knowledge.

Table 2, in Appendix C, shows the time measurements for both microcontrollers. "Software LIKE" corresponds to a software-based implementation, with all cryptographic operations performed in software, without any hardware acceleration. "Hardware LIKE" corresponds to a hardware-accelerated implementation where elliptic curve operations (point addition and point multiplication), field operations (modular multiplication and modular addition) and hash operations are performed using hardware acceleration.

Table 2 shows that the implementation of LIKE comes with a significant overhead, mainly due to the expensive ECC point multiplications and additions which figure in the SoK. We note an (expected) effect of the clock frequency and processor architecture on performance: the software-based implementation on Cortex M4F is comparable with the hardware-based implementation on Cortex M33. Finally, we observe that hardware acceleration substantially reduces the execution time differences observed between base EDHOC and LI-EDHOC for the Cortex M33 device, making lawful-interception more practical for deployment in constrained devices. Future iterations of this implementation could implement hardware acceleration as well in the nRF52840 (Cortex M4F) to further improve performance and reduce computation times.

Moreover, it would be interesting to run future experiments on hardware dedicated for ECC computations, such as [14]. Finally note that, whereas static authentication requires a lengthy signature of knowledge, complexity drops for authentication methods relying only on ephemeral secrets (*i.e.,* where peers use signatures to authenticate).

7 Conclusion

Mobile environments come with a compulsory Lawful Interception (LI) clause for mobile network operators (MNOs). Naïve LI, however, is intrusive, essentially forcing the operator to eavesdrop on all communications. Our work provides *fine-grained, session-specific* LI for the complex EDHOC protocol, which is being standardized for mobile use in the LAKE IETF working group. Our LI-EDHOC scheme provably guarantees key-security, non-frameability, and a proof of LI key-recovery for EDHOC's 4 authentication methods, while preserving identity-protection and explicit authentication. Evaluations on two different ARM Cortex-M platforms, show that our protocol remains efficient in constrained environments, especially when the signatures of knowledge benefit from hardware acceleration. Even better performances are likely to be obtained for signature-based authentication methods, or if the processor were optimized for ECC computations.

Acknowledgement

This work has been partially supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009) and through the PRIvacy-preserving tools for the VAliation and SecurIty of Queries (PRIVASIQ) project (ANR-23-CE39-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

References

1. Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats. *Comms. of the ACM*, 58(10):24–26, 2015.
2. Ghada Arfaoui, Olivier Blazy, Xavier Bultel, Pierre-Alain Fouque, Thibaut Jacques, Adina Nedelcu, and Cristina Onete. How to (legally) keep secrets from mobile operators. In *Proceedings of ESORICS*, pages 23–43, 2021.
3. Mihir Bellare and Ronald L. Rivest. Translucent cryptography—an alternative to key escrow, and its implementation via fractional oblivious transfer. *Journal of Cryptology*, 12(2):117–139, 1999.
4. Felipe Boeira, Mikael Asplund, and Marinho Barcellos. Provable non-frameability for 5G lawful interception. In *Proceedings of WiSec*, page 109–120, 2023.
5. Xavier Bultel and Cristina Onete. Pairing-free secure-channel establishment in mobile networks with fine-grained lawful interception. In *Proceedings of ACM SAC*, page 968–970, 2022.
6. Global Encryption Coalition. Joint statement on the dangers of the may 2024 council of the EU comrpomise proposal on EU CSAM, 2024.
7. Baptiste Cottier and David Pointcheval. Security analysis of the edhoc protocol, 2022.
8. Baptiste Cottier and David Pointcheval. Security analysis of improved edhoc protocol. In *Foundations and Practice of Security*, 2023.
9. EU Council. Resolution on security through encryption and security despite encryption, 2020.
10. FairTrials. Short update: Police in germany defend the use of contact tracing for criminal investigations, 2020.
11. Lorenzo Franceschi-Bicchierai. The 10 biggest revelations from edward snowden's leaks, 2014.
12. Felix Günther and Marc Ilunga Tshibumbu Mukendi. Careful with mac-then-sign: A computational analysis of the edhoc lightweight authenticated key exchange protocol. In *IEEE (EuroS&P)*, pages 773–796, 2023.
13. Xu He, Lixiang Li, and Haipeng Peng. A key escrow-free kp-abe scheme and its application in standalone authentication in iot. *IEEE Internet of Things Journal*, 11(7):11381–11394, 2024.
14. Xianghong Hu, Xueming Li, Xin Zheng, Yuan Liu, and Xiaoming Xiong. A high speed processor for elliptic curve cryptography over nist prime field. *IET Circuits, Devices & Systems*, 16:350–359, 02 2022.
15. Meijuan Huang, Yutian Liu, Bo Yang, Yanqi Zhao, and Mingrui Zhang. Efficient revocable attribute-based encryption with data integrity and key escrow-free. *Information*, 15:32, 01 2024.

16. Ingrid Huso, Marco Olivieri, Leonardo Galgano, Adnan Rashid, Giuseppe Piro, and Gennaro Boggia. Design and implementation of a looking-forward lawful interception architecture for future mobile communication systems. *Comput. Networks*, 249:110518, 2024.
17. Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. A comprehensive, formal and automated analysis of the EDHOC protocol. In *USENIX Security 23*, pages 5881–5898, 2023.
18. Jake Januzelli and Jiayu Xu. A complete characterization of one-more assumptions in the algebraic group model. Cryptology ePrint Archive, Paper 2024/1954, 2024.
19. Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425, 2003.
20. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.
21. Pascal Lafourcade, Elsa López Pérez, Charles Olivier-Anclin, Cristina Onete, Clément Papon, and Mališa Vučinić. Fine-grained, privacy-augmenting li-compliance in the lake standard (full version). `https://hal.science/hal-05126079`, 2025.
22. Elsa Lopez Perez, Thomas Watteyne, Rafael Marin-Lopez, Cristina Onete, and Mališa Vučinić. Pre-shared key authentication with edhoc: The security-performance tradeoff, 2025. awaiting for publication.
23. Elsa López Pérez, Inria Göran Selander, John Preuß Mattsson, Thomas Watteyne, and Mališa Vučinić. Edhoc is a new security handshake standard: An overview of security analysis, 2024.
24. 379 scientists and researchers from 36 countries. Open letter on the position of scientists and researchers on the updated version of the EU's proposed Child Sexual Abuse Regulation, 2024.
25. G. Selander, J. Preuss Mattsson, and F. Palombini. Ephemeral Diffie-Hellman over COSE (EDHOC). RFC 9528, 2024.
26. Giuseppe Ungaro, Francesco Ricchitelli, Ingrid Huso, Giuseppe Piro, and Gennaro Boggia. Design and implementation of a lawful interception architecture for b5g systems based on key escrow. In *IEEE CSCN*, pages 207–207, 2022.
27. Charles Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *Proceedings of EuroS&P*, pages 288–306, 2018.

A NIPoK/SoK instantiation

We present here the instantiation of the SoK used in method 3, *i.e.,* the one we implemented, and describe how to adapt it to method 2 (presented in the core of the paper). LI-EDHOC in method 3 requires, for the SoK, a proof of knowledge of the equality of three discrete logarithms, and of the equality of two other discrete logarithms, in order to certify that the elements $\mathsf{H_d}$, $\mathsf{H_d^1}$ and $\mathsf{H_d^2}$ are well formed. We consider a hash function $\mathcal{H}_{\mathsf{SoK}} : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p^*$ modeled as a random oracle. We use the following instantiation for the $\mathsf{Initiator}$ SoK. The $\mathsf{Initiator}$ has witnesses x and i to the statement $\mathsf{stm_I} = \big[X = g^x \wedge \mathsf{H_I} = (\mathsf{h_I} g^y)^x \wedge \mathsf{H_I^1} = (\mathsf{h_I} g^r)^x \wedge X_1 = g^i \wedge \mathsf{H_I^2} = (\mathsf{h_I} g^y)^i\big]$, and will sign message $m_\mathsf{I} = \omega_\mathsf{I} \| \mathsf{ID_R} \| \mathsf{TH_2} \| \mathsf{TH_3} \| \mathsf{TH_4}$. The $\mathsf{Initiator}$ runs the SoK algorithm as follows and when he checks $\mathsf{ni_I}$, the proxy $\mathsf{Prox_I}$ runs the SoKver algorithm.

$\underline{\mathsf{SoK}(m_\mathsf{I},(x,i),\mathsf{stm}_\mathsf{I})}$: Pick random $r_\mathsf{I}, s_\mathsf{I} \leftarrow\!\!\$\ \mathbb{Z}_p^*$. Compute $I_1 \leftarrow g^{r_\mathsf{I}}$, $I_2 \leftarrow \left(\mathsf{h}_\mathsf{I} g^y\right)^{r_\mathsf{I}}$, $I_3 \leftarrow \left(\mathsf{h}_\mathsf{I} g^r\right)^{r_\mathsf{I}}$, $I_4 \leftarrow g^{s_\mathsf{I}}$, $I_5 \leftarrow \left(\mathsf{h}_\mathsf{I} g^y\right)^{s_\mathsf{I}}$, $\alpha_\mathsf{I} \leftarrow \mathcal{H}_\mathsf{SoK}(I_1 I_2 I_3 I_4 I_5, m_\mathsf{I})$, $\beta_\mathsf{I} \leftarrow r_\mathsf{I} - \alpha_\mathsf{I} x \pmod p$ and $\gamma_\mathsf{I} \leftarrow s_\mathsf{I} - \alpha_\mathsf{I} i \pmod p$. Return $\mathsf{ni}_\mathsf{I} = (\alpha_\mathsf{I}, \beta_\mathsf{I}, \gamma_\mathsf{I})$.

$\underline{\mathsf{SoKver}(m_\mathsf{I},\mathsf{stm}_\mathsf{I},\mathsf{ni}_\mathsf{I})}$: Compute $I_1' \leftarrow g^{\beta_\mathsf{I}}(g^x)^{\alpha_\mathsf{I}}$, $I_2' \leftarrow (\mathsf{h}_\mathsf{I} g^y)^{\beta_\mathsf{I}}(\mathsf{h}_\mathsf{I}^x g^{xy})^{\alpha_\mathsf{I}} = (\mathsf{h}_\mathsf{I} g^y)^{\beta_\mathsf{I}}(\mathsf{H}_\mathsf{I})^{\alpha_\mathsf{I}}$, $I_3' \leftarrow (\mathsf{h}_\mathsf{I} g^r)^{\beta_\mathsf{I}}(\mathsf{h}_\mathsf{I}^x g^{xr})^{\alpha_\mathsf{I}} = (\mathsf{h}_\mathsf{I} g^r)^{\beta_\mathsf{I}}(\mathsf{H}_\mathsf{I}^1)^{\alpha_\mathsf{I}}$, $I_4' \leftarrow g^{\gamma_\mathsf{I}}(g^i)^{\alpha_\mathsf{I}}$ and $I_5' \leftarrow (\mathsf{h}_\mathsf{I} g^y)^{\gamma_\mathsf{I}}(\mathsf{h}_\mathsf{I}^i g^{iy})^{\alpha_\mathsf{I}} = (\mathsf{h}_\mathsf{I} g^y)^{\gamma_\mathsf{I}}(\mathsf{H}_\mathsf{I}^2)^{\alpha_\mathsf{I}}$. Return 1 if $\alpha_\mathsf{I} = \mathcal{H}_\mathsf{SoK}(I_1' I_2' I_3' I_4' I_5', \omega_\mathsf{I})$, else return 0.

On their side, the Responder and his proxy Prox_R work with the statement $\mathsf{stm}_\mathsf{R} = \left[Y = g^y \wedge \mathsf{H}_\mathsf{R} = (\mathsf{h}_\mathsf{R} g^x)^y \wedge \mathsf{H}_\mathsf{R}^1 = (\mathsf{h}_\mathsf{R} g^i)^y \wedge Y_1 = g^r \wedge \mathsf{H}_\mathsf{R}^2 = (\mathsf{h}_\mathsf{R} g^x)^r\right]$, the witnesses y and r, and the message $m_\mathsf{R} = \omega_\mathsf{R} \| \mathsf{ID}_\mathsf{I} \| \mathsf{TH}_2 \| \mathsf{TH}_3 \| \mathsf{TH}_4$. The SoK and SoKver algorithms work in the same way.

In authentication method 2, this SoK is the same on the Initiator side, see Figure 4. On the Responder side, however, the SoK is less computationally expensive, as the Responder proves the equality of two discrete logarithms instead of three, omitting thus the Responder-side equivalents of I_4 and I_5.

B Further properties: identity-protection

Identity-Protection was defined in [8] and tailored to the EDHOC protocol. We slightly modify this game in order to tailor it to our modification of EDHOC. We prove that our protocol preserves the following two properties also achieved by the original scheme: that the Initiator's identity is hidden by encryption against an active attacker; and that the Responder's identity is also hidden by encryption against a passive attacker.

C Additional figures and tables

Operation	**Cortex M4F @ 64 MHz**		**Cortex M33 @ 32 MHz**			
	SW LIKE	**SW base**	**SW LIKE**	**SW base**	**HW LIKE**	**HW base**
Precomputation Phase						
Precomputation (vok_log_auth)	1.8095	-	7.1282	-	0.5798	-
Precomputation (h)	0.0015	-	0.0057	-	0.00005	-
Precomputation (w)	0.0015	-	0.0055	-	0.0009	-
Subtotal	**1.8125**	**-**	**7.1394**	**-**	**0.5808**	**-**
Initiator						
Message_1	0.0016	0.0016	0.0062	0.0062	0.0016	0.0016
Message_3	9.1231	2.8495	35.9294	11.2157	4.1256	1.2649
(SoK)	(6.2477)	-	(24.6097)	-	(2.8484)	-
Subtotal	**9.1247**	**2.8511**	**35.9356**	**11.2219**	**4.1272**	**1.2665**

Table 2: LI-EDHOC handshake (LIKE) vs. EDHOC (base) runtime evaluation (in seconds), for implementations fully run in software (SW) or featuring ECC, field operations, and hashes that are hardware accelerated (HW).

```
$\texttt{Prox}_{\texttt{I/R}}\texttt{Run1}(M_1)$
1 : Parse $M_1$ as $(\mathsf{MCS}\|g^x\|\mathsf{C_I}\|\mathsf{EAD_1})$
2 : $\tau_{\mathsf{I/R}} \leftarrow \omega_{\mathsf{I/R}}\|\mathsf{MCS}\|g^x$    // Initialize $\tau_{\mathsf{I}}/\tau_{\mathsf{R}}$
3 : return $M_1$
```

```
$\texttt{Prox}_{\texttt{I/R}}\texttt{Run2}(M_2)$
1 : Parse $M_2$ as $(g^y\|c_2)$
2 : $\tau_{\mathsf{I/R}} \leftarrow \tau_{\mathsf{I/R}}\|g^y$    // Update $\tau_{\mathsf{I}}/\tau_{\mathsf{R}}$
3 : return $M_2$
```

```
$\texttt{Prox}_{\texttt{I}}\texttt{Run3}(M_3)$
1 : Parse $M_3$ as $(c_3\|\mathsf{H_I}\|\mathsf{H_I^1}\|\mathsf{ni_I}\|S_\mathsf{I})$
    // Decipher $S_\mathsf{I}$ and verify transcript hashes
2 : $K_\mathsf{I} \leftarrow \mathsf{Dec}''(\mathsf{pcsk}_{\mathsf{I},\mathsf{Prox_I}}, S_\mathsf{I})$, parse $K_\mathsf{I}$ as $(\mathsf{sk}_2\|\mathsf{sk}_3\|\mathsf{IV}_3\|t_2)$
3 : $m_2 \leftarrow \mathsf{Dec}(\mathsf{sk}_2, c_2)$, get $\mathsf{ID_R}, \sigma_2$ from $m_2$
4 : $\mathsf{TH}_2 \leftarrow \mathcal{H}(g^y, \mathcal{H}(M_1))$, $\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2, \mathsf{ID_R})$
5 : $\mathsf{ad}_3 \leftarrow (l_{\mathsf{aead}}\|\text{“ ”}\|\mathsf{TH}_3)$, $m_3 \leftarrow \mathsf{Dec}'(\mathsf{sk}_3, \mathsf{IV}_3, \mathsf{ad}_3, c_3)$
6 : $\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3, \mathsf{ID_I})$
    // Verify SoK and HMAC, update $\tau_\mathsf{I}$
7 : if $\mathsf{SoKver}(\omega_\mathsf{I}\|\mathsf{ID_R}\|\mathsf{TH}_2\|\mathsf{TH}_3\|\mathsf{TH}_4, \mathsf{stm_I}, \mathsf{ni_I}) \neq 1$
8 :   or if $\mathsf{DS}.\texttt{Verif}(\mathsf{spk_R}, (l_{\mathsf{sig}}\|\mathsf{CTX}_2\|t_2), \sigma_2) \neq 1$
9 :     then return $\perp$
10 : $\tau_\mathsf{I} \leftarrow \tau_\mathsf{I}\|g^i\|\mathsf{TH}_2\|\mathsf{TH}_3\|\mathsf{TH}_4\|M_1\|m_2\|m_3\|\mathsf{H_I}\|\mathsf{H_I^1}\|\mathsf{ni_I}\|\mathsf{ID_R}\|t_2$
11 : return $M_3' \leftarrow c_3$
```

```
$\texttt{Prox}_{\texttt{R}}\texttt{Run3}(M_4)$
1 : Parse $M_4$ as $(c_4\|n_r\|\mathsf{val}\|\mathsf{TH}_5\|\mathsf{H_R}\|\mathsf{H_R^1}\|\mathsf{ni_R}\|S_\mathsf{R})$
    // Decipher $S_\mathsf{R}$ and verify transcript hashes
2 : $K_\mathsf{R} \leftarrow \mathsf{Dec}''(\mathsf{pcsk}_{\mathsf{R},\mathsf{Prox_R}}, S_\mathsf{R})$
3 : Parse $K_\mathsf{R}$ as $(\mathsf{sk}_2\|\mathsf{sk}_3\|\mathsf{IV}_3\|t_2)$
4 : $m_2 \leftarrow \mathsf{Dec}(\mathsf{sk}_2, c_2)$, $\mathsf{TH}_2 \leftarrow \mathcal{H}(g^y, \mathcal{H}(M_1))$
5 : $\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2, \mathsf{ID_R})$, $\mathsf{ad}_3 \leftarrow (l_{\mathsf{aead}}\|\text{“ ”}\|\mathsf{TH}_3)$
6 : $m_3 \leftarrow \mathsf{Dec}'(\mathsf{sk}_3, \mathsf{IV}_3, \mathsf{ad}_3, c_3)$, get $\mathsf{ID_I}$ from $m_3$
7 : $\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3, \mathsf{ID_I})$
    // Verify SoK and HMAC, update and sign $\tau_\mathsf{R}$
8 : if $\mathsf{DS}.\texttt{Verif}(\mathsf{spk_R}, (l_{\mathsf{sig}}\|\mathsf{CTX}_2\|t_2), \sigma_2) \neq 1$
9 :   then return $\perp$
10 : if $\mathsf{SoKver}(\omega_\mathsf{R}\|\mathsf{ID_I}\|\mathsf{TH}_2\|\mathsf{TH}_3\|\mathsf{TH}_4, \mathsf{stm_R}, \mathsf{ni_R}) \neq 1$
11 :   then return $\perp$
12 : if $\mathsf{TH}_5 \neq \mathcal{H}(\mathsf{TH}_4, \mathsf{val}, n_r)$ then return $\perp$
13 : $\tau_\mathsf{R} \leftarrow \tau_\mathsf{R}\|g^i\|\mathsf{TH}_2\|\mathsf{TH}_3\|\mathsf{TH}_4\|M_1\|m_2\|m_3\|$
       $\mathsf{H_R}\|\mathsf{H_R^1}\|\mathsf{ni_R}\|\mathsf{ID_I}\|t_2\|n_r\|\mathsf{val}$
14 : $\sigma_\mathsf{R} \leftarrow \mathsf{DS}.\texttt{Sign}(\mathsf{ssk}_{\mathsf{Prox_R}}, \tau_\mathsf{R})$, $\mathsf{sst_R} \leftarrow \tau_\mathsf{R}\|\sigma_\mathsf{R}$
15 : return $M_4' \leftarrow (c_4\|n_r\|\mathsf{val}\|\mathsf{TH}_5)$
```

```
$\texttt{Prox}_{\texttt{I}}\texttt{Run4}(M_4')$
1 : Parse $M_4'$ as $(c_4\|n_r\|\mathsf{val}\|\mathsf{TH}_5)$
2 : if $\mathsf{TH}_5 = \mathcal{H}(\mathsf{TH}_4, \mathsf{val}, n_r)$ then return $\perp$
    // Update and sign $\tau_\mathsf{I}$
3 : $\tau_\mathsf{I} \leftarrow \tau_\mathsf{I}\|n_r\|\mathsf{val}$, $\sigma_\mathsf{I} \leftarrow \mathsf{DS}.\texttt{Sign}(\mathsf{ssk}_{\mathsf{Prox_I}}, \tau_\mathsf{I})$
4 : $\mathsf{sst_I} \leftarrow \tau_\mathsf{I}\|\sigma_\mathsf{I}$, return $M_4'' \leftarrow M_4'$
```

Fig. 5: Proxies instantiation of the LI-EDHOC protocol with ID = 2.

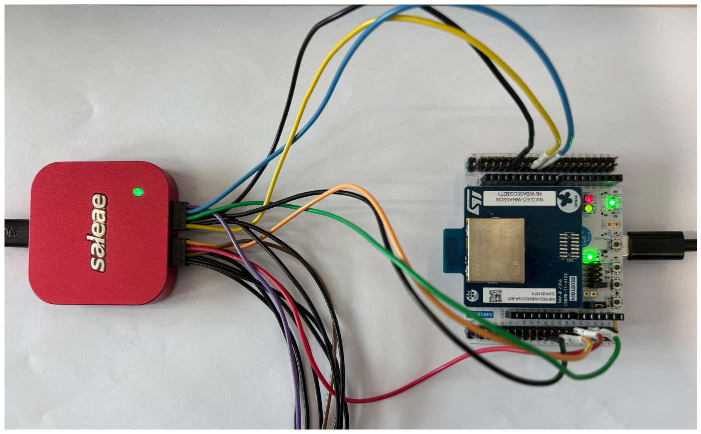

Fig. 6: Our evaluation setup for the embedded evaluation board STM32WBA55CG (to the right). We toggle General Purpose Input/Output pins and record execution times using a logic analyzer (to the left) which runs on a PC.