Security Analysis of the EDHOC protocol

Baptiste Cottier, David Pointcheval

▶ **To cite this version:**

Baptiste Cottier, David Pointcheval. Security Analysis of the EDHOC protocol. 2022. hal-03772082v1

HAL Id: hal-03772082
https://hal.science/hal-03772082v1

Preprint submitted on 7 Sep 2022 (v1), last revised 23 Nov 2022 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Security Analysis of the EDHOC protocol

Baptiste Cottier and David Pointcheval

DIENS, École normale supérieure, CNRS, Inria, PSL University, Paris, France

Abstract. Ephemeral Diffie-Hellman Over COSE (EDHOC) aims at being a very compact and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys. It is expected to provide mutual authentication, forward secrecy, and identity protection, with a 128-bit security level.
A formal analysis has already been proposed at SECRYPT '21, on a former version, leading to some improvements, in the ongoing evaluation process by IETF. Unfortunately, while formal analysis can detect some misconceptions in the protocol, it cannot evaluate the actual security level.
In this paper, we study the last version. Without complete breaks, we anyway exhibit attacks in 2^{64} operations, which contradict the expected 128-bit security level. We thereafter propose improvements, some of them being at no additional cost, to achieve 128-bit security for all the security properties (i.e. key privacy, mutual authentication, and identity-protection).

1 Protocol Description

Ephemeral Diffie-Hellman over COSE [Sch16] (EDHOC) aims to provide a common session key to two parties potentially running on constrained devices over low-power IoT radio communication technologies. EDHOC protocol can be instantiated with several settings:

- *Authentication Method*: Each party (Initiator and Responder) can use an authentication method: either with a signature scheme (`SIG`), or with a static Diffie-Hellman key (`STAT`).

Value	Initiator	Responder
0	`SIG`: Signature	`SIG`: Signature
1	`SIG`: Signature	`STAT`: Static DH
2	`STAT`: Static DH	`SIG`: Signature
3	`STAT`: Static DH	`STAT`: Static DH

- *Cipher Suites*: Ordered set of protocol security settings. Initial paper offers many possible suites, but we focus on the most aggressive cipher suites setting the MAC length to 8 bytes, i.e. Cipher Suites 0 and 2 which share the following parameters:

(Application) AEAD	Hash	MAC len
AES-CCM-16-64-128	SHA-256	8

The difference between Cipher Suites 0 and 2 is the Elliptic Curve used: X25519 in suite 0 and P-256 in suite 2.
- *Connection Identifiers.* Data that may be used to correlate between messages and facilitate retrieval of protocol state in EDHOC and application.
- *Credentials and Identifiers.* They are used to identify and optionally transport the authentication keys of the Initiator and the Responder.

We suppose both the Initiator and the Responder are aware that the authentication method is 3, the `STAT`/`STAT`. Also, as said before, the difference between Cipher Suite 0 and Cipher Suite 2 is the choice of the Elliptic Curve. As both curves provide the same security guarantee and are more an implementation concern, we do not include the Cipher Suite ID `Suites_I` (as it appears in [SMP22]) in the first message of the protocol.

Extract and Expand. In the EDHOC Key-Schedule defined below, the pseudo-random keys (PRK) are derived using an extraction function. In our context, as we study cipher suites 0 and 2, our hash algorithm is set as SHA-256. Therefore $\mathsf{Extract}(\mathsf{salt}, \mathsf{IKM}) = \mathsf{HKDF\text{-}Extract}(\mathsf{salt}, \mathsf{IKM})$ is defined with SHA-256, where IKM holds for Input Keying Material (in our context, this will be some Diffie-Hellman keys) and $\mathsf{Expand}(\mathsf{PRK}, \mathsf{info}, \mathsf{len}) = \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}, \mathsf{info}, \mathsf{len})$ where info contains the transcript hash (TH_2, TH_3 or TH_4), the name of the derived key and some context, while len denotes the output length.

Key Schedule. During the key exchange, several cryptographic computations occur. The keying material, including MAC Keys (as we study `STAT`/`STAT` method), Encryption Keys and Initialisation Vectors result from a key schedule, adapted from Norrman et al. [**?**]. In Fig. 1, purple boxes denote Diffie-Hellman shared secrets ($g^{x_e y_e}, g^{y_s x_e}, g^{x_s y_e}$), where x_e and y_e denote the ephemeral DH keys, while x_s and y_s are the long-term static DH keys. From those Diffie-Hellman shared secrets, we extract the pseudo random keys $\mathsf{PRK}_{2e}, \mathsf{PRK}_{3e2m}$ and PRK_{4e3m}, in light blue. Those keys are then expanded to compute the encryption material (keys and IV), in red, and the authentication tags t_2 and t_3, in yellow. Keys PRK_{2e} and PRK_{3e2m} are also used to compute the salts salt_{3e2m} and salt_{4e3m} respectively, in grey, used in the HKDF-Extract function. The final session key $\mathsf{PRK}_{\mathsf{out}}$, backgrounded in green, is computed calling HKDF-Expand on PRK_{4e3m}. Transcript hashes, denoted TH_i, are used as input to the HKDF-Expand function. More precisely, with SHA-256 as $\mathcal{H}$, we have:

$$\mathsf{TH}_2 = \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1)) \qquad \mathsf{TH}_3 = \mathcal{H}(\mathsf{TH}_2, m_2) \qquad \mathsf{TH}_4 = \mathcal{H}(\mathsf{TH}_3, m_3)$$

where m_1 is the first message sent by the Initiator, m_2 and m_3 respectively are the plaintexts respectively encrypted in the message 2 and message 3.

Authenticated Encryption with Associated Data (AEAD). As said above, the cipher suites we work on both use AES-CCM-16-64-128. We detail this Authenticated Encryption scheme:

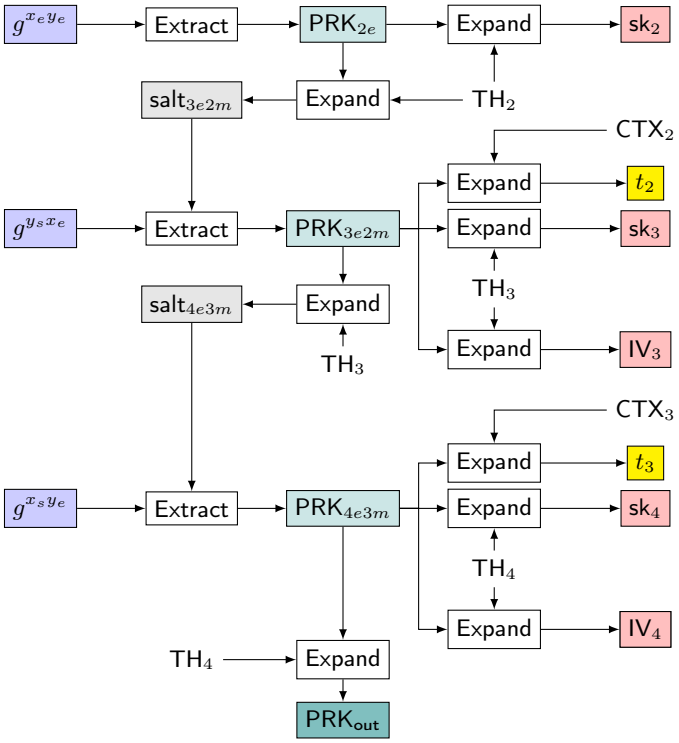

Fig. 1. Key Derivation (for the `STAT-STAT` Method) from [NSB21].

- AES-CCM: CCM, for Counter with CBC-MAC is an AES mode providing both encryption and authentication. CCM mode combines the CBC-MAC and the CTR (counter) mode of encryption. The first step consists in calculating a tag T, then, encrypt the message and the tag using the counter mode.
- 16: messages length is limited to 2^{16} bytes long (64KiB). Therefore, the nonce is 13 bytes long allowing $2^{13\times 8}$ possibles values of the nonce without repeating
- 64: Tag is 64 bits long.
- 128: Key is 128 bits long.

Connection Identifiers (from [SMP22]). Connection identifiers ($\mathsf{C_I}$ and $\mathsf{C_R}$) may be used to correlate EDHOC messages and facilitate the retrieval of protocol state during EDHOC protocol execution or in a subsequent application protocol. The connection identifiers do not have any cryptographic purpose in EDHOC.

EDHOC-Exporter and EDHOC-KeyUpdate. At the end of the protocol, the Initiator and the Responder compute TH_4. This value can be used in case an application need to export the EDHOC session key. Also, in case the key needs to be

updated, the Initiator and the Responder rerun $\mathsf{HKDF\text{-}Extract}$, with PRK_{4e3m} as input, together with a random nonce agreed upon by the Initiator and the Responder.

Protocol. The detailed description of the protocol is given in Fig 3. The final session key is $\mathtt{SK} = \mathsf{PRK}_{\mathsf{out}}$

$\mathbb{G} =< g >$	Cyclic group generated by g
p	Size of the group $\mathbb{G}$
$\mathcal{H}$	Hash function SHA-256 (256 bits digest)
X_e, x_e	Initiator Ephemeral DH Public and Secret Key
X_s, x_s	Initiator Static DH Public and Secret Key
Y_e, y_e	Responder Ephemeral DH Public and Secret Key
Y_s, y_s	Responder Static DH Public and Secret Key
EAD	External Authorized Data
sk	Secret key
$\mathcal{E}, \mathcal{D}$	One-time Encryption and Decryption
$\mathcal{E}', \mathcal{D}'$	Authenticated Encryption and Decryption with Associated Data
$\perp$	Protocol abortion
TH	Transcript Hash
t_2, t_3	MAC tags
$\mathsf{C_R}, \mathsf{C_I}$	Connection Identifiers
ℓ_{mac}	MAC output length
ℓ_2	Plaintext length of the first message send by the responder
ℓ_{hash}	Hash length
$\ell_{\mathsf{key}}, \ell_{\mathsf{iv}}$	Key and IV length

Fig. 2. Notations

2 Security Concerns

Security Goals. The security goals of an authenticated key exchange protocol are:

- *Key Privacy*: Equivalent to Implicit Authentication. At most both participants know the final session key, which should remain indistinguishable from random to outsiders. With additional *Perfect Forward Secrecy*, by compromising the long-term credential of either peer, an attacker shall not be able to distinghuish past session keys from random keys. In our context, this will rely on a Diffie-Hellman assumption.
- *Mutual Authentication*: Equivalent to explicit authentication. Exactly both participants have the material to compute the final session key.
- *Identity Protection*: At most both participants know the identity of the Initiator and the Responder.

Initiator		Responder
$X_s = g^{x_s}$	static keys	$Y_s = g^{y_s}$
$\underline{\mathsf{InitRun1}(\mathsf{ID_I})}$		
$x_e \xleftarrow{\$} \mathbb{Z}_p, X_e \leftarrow g^{x_e}$		
$\mathsf{C_I} \xleftarrow{\$} \{0,1\}^{nl}$		
$m_1 \leftarrow (X_e \Vert \mathsf{C_I} \Vert \mathsf{EAD}_1)$	$\xrightarrow{m_1}$	$\underline{\mathsf{RespRun1}(\mathsf{ID_R}, y_s, m_1)}$
		Parse m_1 as $(X_e \Vert c \Vert \mathsf{EAD}_1)$
		$y_e \xleftarrow{\$} \mathbb{Z}_p, Y_e \leftarrow g^{y_e}$
		$\mathsf{C_R} \xleftarrow{\$} \{0,1\}^{nl}$
		$\mathsf{sid} \leftarrow (\mathsf{C_I}, \mathsf{C_R}, X_e, Y_e)$
		$\mathsf{PRK}_{2e} \leftarrow \mathsf{HKDF\text{-}Extract}(\text{""}, {X_e}^{y_e})$
		$\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1))$
		$\mathsf{sk}_2 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$
		$\mathsf{salt}_{3e2m} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
		$\mathsf{PRK}_{3e2m} \leftarrow \mathsf{HKDF\text{-}Extract}(\mathsf{salt}_{3e2m}, {X_e}^{y_s})$
		$\mathsf{CTX}_2 \leftarrow (\mathsf{ID_R} \Vert \mathsf{TH}_2 \Vert Y_s \Vert \mathsf{EAD}_2)$
		$t_2 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$
		$m_2 \leftarrow (\mathsf{ID_R} \Vert t_2 \Vert \mathsf{EAD}_2)$
$\underline{\mathsf{InitRun2}(\mathsf{ID_I}, x_s, Y_s, (Y_e, c_2, \mathsf{C_R}))}$	$\xleftarrow{Y_e, c_2, \mathsf{C_R}}$	$c_2 \leftarrow \mathcal{E}(\mathsf{sk}_2, m_2)$
$\mathsf{PRK}_{2e} \leftarrow \mathsf{HKDF\text{-}Extract}(\text{""}, {Y_e}^{x_e})$		
$\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1))$		
$\mathsf{sk}_2 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$		
Set $m_2 \leftarrow \mathcal{D}(\mathsf{sk}_2, c_2)$		
Parse m_2 as $(\mathsf{ID_R} \Vert t_2 \Vert \mathsf{EAD}_2)$		
$\mathsf{CTX}_2 \leftarrow (\mathsf{ID_R} \Vert \mathsf{TH}_2 \Vert Y_s \Vert \mathsf{EAD}_2)$		
$\mathsf{salt}_{3e2m} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$		
$\mathsf{PRK}_{3e2m} \leftarrow \mathsf{HKDF\text{-}Extract}(\mathsf{salt}_{3e2m}, {Y_s}^{x_e})$		
$t_2' \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$		
if $t_2' \neq t_2$: **return** $\perp$		
$\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$		
$\mathsf{sk}_3 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$		
$\mathsf{IV}_3 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$		
$\mathsf{salt}_{4e3m} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$		
$\mathsf{PRK}_{4e3m} \leftarrow \mathsf{HKDF\text{-}Extract}(\mathsf{salt}_{4e3m}, {Y_e}^{x_s})$		
$\mathsf{accepted} \leftarrow 1$		
$\mathsf{CTX}_3 \leftarrow (\mathsf{ID_I} \Vert \mathsf{TH}_3 \Vert X_s \Vert \mathsf{EAD}_3)$		
$t_3 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$		
$m_3 \leftarrow (\mathsf{ID}_I \Vert t_3 \Vert \mathsf{EAD}_3)$		
$c_3 \leftarrow \mathcal{E}'(\mathsf{sk}_3, \mathsf{IV}_3, m_3)$	$\xrightarrow{c_3}$	$\underline{\mathsf{RespRun2}(\mathsf{ID}, \mathtt{st}, \mathtt{peerpk}, c_3)}$
		$\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$
		$\mathsf{sk}_3 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
		$\mathsf{IV}_3 \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$
		Set $m_3 \leftarrow \mathcal{D}'(\mathsf{sk}_3, \mathsf{IV}_2, c_3)$
		if $m_3 = \perp$: **return** $\perp$
		Parse m_3 as $(\mathsf{ID_I} \Vert t_3 \Vert \mathsf{EAD}_3)$
		$X_s \leftarrow \mathtt{peerpk}[\mathsf{ID_I}]$
		$\mathsf{salt}_{4e3m} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
		$\mathsf{PRK}_{4e3m} \leftarrow \mathsf{HKDF\text{-}Extract}(\mathsf{salt}_{4e3m}, {X_s}^{y_e})$
		$\mathsf{accepted} \leftarrow 1$
		$\mathsf{CTX}_3 \leftarrow (\mathsf{ID_I} \Vert \mathsf{TH}_3 \Vert X_s \Vert \mathsf{EAD}_3)$
		$t_3' \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$
		if $t_3' \neq t_3$: **return** $\perp$
$\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$		$\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$
$\mathsf{PRK_{out}} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$		$\mathsf{PRK_{out}} \leftarrow \mathsf{HKDF\text{-}Expand}(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$
$\mathsf{terminated} \leftarrow 1$		$\mathsf{terminated} \leftarrow 1$
$\mathtt{SK} \leftarrow \mathsf{PRK_{out}}$		$\mathtt{SK} \leftarrow \mathsf{PRK_{out}}$

Fig. 3. EDHOC (draft-ietf-lake-edhoc-15) in the `STAT`/`STAT` Authentication Method

Random Oracle Model. For the security analysis, we model Hash and Key Derivation Functions as random oracles. Respectively, the random oracles RO_T and RO_P will model HKDF-Extract and HKDF-Expand functions as perfect random functions.

Computational Diffie-Hellman Assumption (CDH). The CDH assumption in a group $\mathbb{G}$ states that given g^u and g^v, where u, v were drawn at random from $\mathbb{Z}_p$, it is hard to compute g^{uv}. This can be defined more precisely by considering an Experiment $\mathbf{Exp}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})$, in which we select two values u and v in $\mathbb{Z}_p$, compute $U = g^u$ and $V = g^v$, and then give both U and V to $\mathcal{A}$. Let Z be the output of $\mathcal{A}$. Then, the Experiment $\mathbf{Exp}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})$ outputs 1 if $Z = g^{uv}$ and 0 otherwise. We define the advantage of $\mathcal{A}$ in violating the CDH assumption as $\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A}) = 1]$ and the advantage function of the group, $\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(t)$, as the maximum value of $\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t.

Gap Diffie-Hellman (GDH). The Gap Diffie-Hellman problem aims to solve a CDH instance $(g, U = g^u, V = g^v)$, as above, with access to a Decisional Diffie-Hellman oracle DDH returning 1 if a tuple (g, g^a, g^b, g^c) is a Diffie-Hellman tuple, and 0 otherwise. We define the advantage function of the group $\mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{DDH}})$, as the maximum value of $\mathbf{Adv}^{\mathsf{CDH}}_{\mathbb{G}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t and making at most q_{DDH} queries to the DDH oracle.

Symmetric Encryption. In the following, we will use several symmetric encryption schemes, such as $\Pi = (\mathcal{E}, \mathcal{D})$ with keys in $\mathcal{K}$ and messages in $\mathcal{M}$, with various properties:

Injectivity. $\Pi = (\mathcal{E}, \mathcal{D})$ is injective if

$$\forall k \in \mathcal{K}, \mathcal{E}(k, m_1) = \mathcal{E}(k, m_2) \Longrightarrow m_1 = m_2.$$

Semantic Security. $\Pi = (\mathcal{E}, \mathcal{D})$ is semantically secure if, for chosen messages $m_0, m_1 \in \mathcal{M}$, an adversary cannot distinguish $\mathcal{E}(k, m_0)$ and $\mathcal{E}(k, m_1)$ with a negligible advantage for a random key $k \in \mathcal{K}$. This can be defined more precisely by considering the Experiment $\mathbf{Exp}^{\mathsf{ind}}_{\Pi}(\mathcal{A})$, for indistinguishability, in which $\mathcal{A}$ selects and gives us two messages m_0 and m_1, then we choose $b \in \{0, 1\}$ and $k \in \mathcal{K}$, compute and send $c = \mathcal{E}(k, m_b)$ to $\mathcal{A}$. Let $b' \in \{0, 1\}$ be the output of $\mathcal{A}$. Then, the Experiment $\mathbf{Exp}^{\mathsf{ind}}_{\Pi}(\mathcal{A})$ outputs 1 if $b' = b$ and 0 otherwise. We define the advantage of $\mathcal{A}$ in violating the semantic security of Π as $\mathbf{Adv}^{\mathsf{ind}}_{\Pi}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{ind}}_{\Pi}(\mathcal{A}) = 1]$ and the advantage function $\mathbf{Adv}^{\mathsf{ind}}_{\Pi}(t)$, as the maximum value of $\mathbf{Adv}^{\mathsf{ind}}_{\Pi}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t.

Authenticated Encryption (with Associated Data). In addition, to semantic security (possibly with access to an encryption/decryption oracle), we require an unforgeability property ($\mathsf{uf} - \mathsf{cma}$, for Unforgeability under Chosen Message Attacks). More precisely, let $\Pi = (\mathcal{E}, \mathcal{D})$ be an Authenticated Encryption scheme. Consider the Experiment $\mathbf{Exp}^{\mathsf{uf-cma}}_{\Pi}(\mathcal{A})$ in which $\mathcal{A}$ is given access to an encryption oracle $\mathcal{E}(k, \cdot)$ and a decryption oracle $\mathcal{D}(k, \cdot)$, for a random key $k \in \mathcal{K}$. The Experiment returns 1 if $\mathcal{A}$ outputs a valid ciphertext

c, which means that $\mathcal{D}(k, c) \neq \perp$ while c has not been obtained as the output of an encryption query to $\mathcal{E}(k, \cdot)$. We define the forger's advantage of $\mathcal{A}$ as $\mathbf{Adv}_{\Pi}^{\mathsf{uf-cma}}(\mathcal{A}) = \Pr[\mathbf{Exp}_{\Pi}^{\mathsf{uf-cma}}\mathcal{A} = 1]$ and the advantage function $\mathbf{Adv}_{\Pi}^{\mathsf{uf-cma}}(t)$ as the maximum value of $\mathbf{Adv}_{\Pi}^{\mathsf{uf-cma}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t.

In the above game of basic semantic security, the adversary has no access to any encryption/decryption oracle, which is a very weak security notion, also known as one-time privacy, as the key is used once only. The One-Time Pad satisfies this property. Note that the One-Time Pad is also injective.

3 Key Privacy

An authenticated key exchange protocol AKE can be defined using three algorithms:

- KeyGen(ID) takes an identity ID as input and samples a long term pair of keys (pk, sk). Key pairs are associated to that user with identity ID, and the public key is added to the list `peerpk`.
- Activate(ID, `role`) takes as input a user identity ID and its `role` $\in$ {initiator, responder}. Activate returns a state st and a message m'.
- Run(ID, sk, `peerpk`, m) delivers m to the session of user ID with secret key sk and state `st`. Run update the state `st` and returns the response message m'

Algorithm Run takes as implicit argument a state `st`, that contains some informations, denoted in `typewriter font`, about the actual session. A value `peerid` $\in \mathbb{N}$ used to identify the intended partner identity of the session, a `role`, either initiator or responder, defining the role played by the session. The state also contains the `status` of the actual session. Either the session `status` is running, meaning the session has been activated, accepted, meaning that a party has all the material to compute the session key or terminated when the session key is computed and the protocol is ended. We also consider a rejected flag in case the session meets a mistake in the verification phase. The final session key is stored in `SK` and set as $\perp$ until defined by the key schedule. Finally, `sid` stores the session identifier used to define partnered session in the security model.

We describe in Figure 4 the security game introduced in [DG20] following the framework by Bellare *et al.* [BR06]. After initializing the game, the adversary $\mathcal{A}$ is given multiple access to the following queries:

- NewUser: Generates a new user by generating a new pair of keys.
- Send: Controls activation and message processing of sessions
- SessionKeyReveal: Reveals the session key of a terminated session.
- LongTermKeyReveal: Corrupts a user and reveals its long term secret key.
- Test: Provides a real-or-random challenge on the session key of the queried session.

Then, the adversary makes a single call to the Finalize algorithm, which returns the result of the predicate $[b' = b]$, where b' is the guess of $\mathcal{A}$ and b is the challenge bit, after succeeding through the following predicates:

Sound: ensures at most two sessions share the same `sid`. Once a couple of sessions is detected, the predicate checks if both of them have their `status` accepted, one session user ID is the `peerid` of the other session, with different `role` and the same final session key `SK`. If one of these property is not verified, the adversary breaks the soundness.

Fresh: detects trivially attacked sessions. First, it ensures that neither the session key is revealed or any of the peers of the session is corrupted before the acceptance time t_{acc}. In a second time, the Fresh predicate ensures that the partnered session is neither tested or revealed. If such a session is detected, we set the answer bit b' as 0.

The advantage of an adversary $\mathcal{A}$ against the key privacy is its bias in guessing b, from the random choice: $\mathbf{Adv}^{\mathsf{kp-ake}}(\mathcal{A}) = \Pr[b' = b] - 1/2$. Therefore, in Figure 5 we give a formalized description of the EDHOC protocol compliant with the security game made in Figure 4. The protocol is analyzed in the random oracle model, therefore, HKDF can be substituted by respective random oracles.

Theorem 1. *The above EDHOC protocol satisfies the key privacy property under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\boldsymbol{Adv}^{kp-ake}_{EDHOC}(t; q_{RO}, n_\sigma, N)$ upper-bounded by*

$$\boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, n_\sigma \cdot q_{RO}) + 2N \cdot \boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, q_{RO}) + \frac{{q_{RO}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_0. This game is the key privacy security game $G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}$ (defined in Figure 4) played by $\mathcal{A}$ using the KeyGen, Activate and Run algorithms (defined in Figure 5). The KeyGen algorithm generates a long term pair of key, calling Activate with an user with identity u, $\mathcal{A}$ creates its i-th session with u, denoted π^i_u.

$$\Pr[\textsc{Succ}_0] = \Pr[G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}],$$

where the event Succ means $b' = b$.
We stress that in this security model, with Perfect Forward Secrecy, we use the weak definition of corruption, meaning that a query to LongTermKeyReveal only reveals the long-term key, while the ephemeral key remains unrevealed. We say a party/session is non-corrupted if no query to LongTermKeyReveal has been made before the time of acceptance t_{acc}, where we consider each block (InitRun1, InitRun2, RespRun1, RespRun2) as atomic. Then corruptions can only happen between two calls to simulated players.

Game G_1. In this game, we simulate the random oracles by lists that are empty at the beginning of the game. As RO_T and $\mathcal{H}$ always return a digest of size ℓ_{hash}, we simply use the simulation oracle SO_T and $\mathsf{SO}_{\mathcal{H}}$ respectively. However, RO_P may return values of several lengths: ℓ_2 for the one-time key encrypting the responder first message, ℓ_{hash} for the salt values and the session key, ℓ_{key} and ℓ_{iv} for the AEAD key length and Initialisation Vector

Initialize ()

$1:$ $\mathsf{time} \leftarrow 0$
$2:$ $\mathsf{users} \leftarrow 0$
$3:$ $b \leftarrow\$ \{0,1\}$

NewUser ()

$1:$ $\mathsf{users} \leftarrow \mathsf{users} + 1$
$2:$ $(\mathsf{pk}_{\mathbf{users}}, \mathsf{sk}_{\mathbf{users}}) \leftarrow\$ \mathsf{KGen}$
$3:$ $\mathsf{revltk}_{\mathbf{users}} \leftarrow \infty$
$4:$ $\mathtt{peerpk}[\mathsf{users}] \leftarrow \mathsf{pk}_{\mathbf{users}}$
$5:$ **return** $\mathsf{pk}_{\mathbf{users}}$

LongTermKeyReveal(u)

$1:$ $\mathsf{time} \leftarrow \mathsf{time} + 1$
$2:$ $\mathsf{revltk}_u \leftarrow \mathsf{time}$
$3:$ **return** sk_u

SessionKeyReveal(u, i)

$1:$ **if** $\pi_u^i = \perp$ or
$\quad \pi_u^i.\mathtt{status} \neq \mathsf{accepted}$:
$2:$ $\quad$ **return** $\perp$
$3:$ $\pi_u^i.revealed \leftarrow \mathbf{true}$
$4:$ **return** $\pi_u^i.\mathtt{SK}$

Finalize(b')

$1:$ **if** $\neg\mathsf{Sound}$:
$2:$ $\quad$ **return** 1
$3:$ **if** $\neg\mathsf{Fresh}$:
$4:$ $\quad b' \leftarrow 0$
$5:$ **return** $[b = b']$

Send(u, i, m)

$1:$ **if** $\pi_u^i = \perp$:
$2:$ $\quad (\mathtt{peerid}, \mathtt{role}) \leftarrow m$
$3:$ $\quad (\pi_u^i, m') \leftarrow\$ \mathsf{Activate}(u, \mathsf{sk}_u, \mathtt{peerid}, \mathtt{peerpk}, \mathtt{role})$
$4:$ $\quad \pi_u^i.t_{acc} \leftarrow 0$
$5:$ **else** :
$6:$ $\quad (\pi_u^i, m') \leftarrow\$ \mathsf{Run}(u, \mathsf{sk}_u, \pi_u^i, \mathtt{peerpk}, \mathtt{role})$
$7:$ **if** $\pi_u^i.\mathtt{status} = \mathsf{accepted}$:
$8:$ $\quad \mathsf{time} \leftarrow \mathsf{time} + 1$
$9:$ $\quad \pi_u^i.t_{acc} \leftarrow \mathsf{time}$
$10:$ **return** m'

Test(u, i)

$1:$ **if** $\pi_u^i = \perp$ or
$\quad \pi_u^i.\mathtt{status} \neq \mathsf{accepted}$ or $\pi_u^i.\mathtt{tested}$
$2:$ $\quad$ **return** $\perp$
$3:$ $\pi_u^i.\mathtt{tested} \leftarrow \mathbf{true}$
$4:$ $T \leftarrow T \cup \{\pi_u^i\}$
$5:$ $k_0 \leftarrow \pi_u^i.\mathtt{SK}$
$6:$ $k_1 \leftarrow\$ \mathsf{KE.KS}$
$7:$ **return** k_b

Sound

$1:$ **if** $\exists$ distinct $\pi_u^i, \pi_v^j, \pi_w^k$ with $\pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid} = \pi_w^k.\mathsf{sid}$:
$2:$ $\quad$ **return false**
$3:$ **if** $\exists \pi_u^i, \pi_v^j$ with
$4:$ $\quad \pi_u^i.\mathtt{status} = \pi_v^j.\mathtt{status} = \mathsf{accepted}$
$5:$ $\quad$ and $\pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid}$
$6:$ $\quad$ and $\pi_u^i.\mathtt{peerid} = u$ and $\pi_v^j.\mathtt{peerid} = v$
$7:$ $\quad$ and $\pi_u^i.\mathtt{role} \neq \pi_v^j.\mathtt{role}$, but $\pi_u^i.\mathtt{SK} \neq \pi_v^j.\mathtt{SK}$
$8:$ $\quad$ **return false**
$9:$ **return true**

Fresh

$1:$ $\forall \pi_u^i \in T$
$2:$ $\quad$ **if** $\pi_u^i.\mathsf{revealed}$
$\quad\quad$ or $\mathsf{revltk}_{\pi_u^i.\mathtt{peerid}} < \pi_u^i.t_{acc}$:
$3:$ $\quad\quad$ **return false**
$4:$ $\quad$ **if** $\exists \pi_v^j \neq \pi_u^i$ **s.t.**
$\quad\quad\quad \pi_u^i.\mathsf{sid} = \pi_u^i.\mathsf{sid}$ and
$\quad\quad\quad (\pi_v^j.\mathsf{tested}$ or $\pi_v^j.\mathsf{revealed})$:
$5:$ $\quad\quad$ **return false**
$6:$ **return true**

Fig. 4. Authenticated Key Exchange Key Privacy Security Game $G^{\mathsf{kp-ake}}_{\mathsf{AKE},\mathcal{A}}$

KeyGen()

$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$
$\mathsf{pk} \leftarrow g^{\mathbb{Z}_p}$
return $(\mathsf{pk}, \mathsf{sk})$

Activate($\mathsf{ID}, role$)

$\mathtt{role} \leftarrow role$
$\mathtt{status} \leftarrow \mathsf{running}$
if $role = \mathsf{initiator}$:
  $m' \leftarrow \mathsf{InitRun1}(\mathsf{ID})$
else $m' \leftarrow \perp$
return m

InitRun1($\mathsf{ID_I}$)

$x_e \xleftarrow{\$} \mathbb{Z}_p$
$X_e \leftarrow g^{x_e}$
$\mathsf{C_I} \xleftarrow{\$} \{0,1\}^{nl}$
$\mathsf{st} \leftarrow (\mathsf{C_I}, X_e, x_e)$
$m_1 \leftarrow (\mathsf{C_I} \| X_e \| \mathsf{EAD}_1)$
return m_1

InitRun2($\mathsf{ID_I}, x_s, (Y_e, c_2, \mathsf{C_R})$)

$\mathsf{PRK}_{2e} \leftarrow \mathsf{RO}_T(\text{""}, {Y_e}^{x_e})$
$\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1))$
$\mathsf{sk}_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$
$m_2 \leftarrow \mathcal{D}(\mathsf{sk}_2, c_2)$
$(\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2) \leftarrow m_2$
$\mathsf{CTX}_2 \leftarrow (\mathsf{ID_R} \| \mathsf{TH}_2 \| Y_s \| \mathsf{EAD}_2)$
$\mathsf{salt}_{3e2m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
$\mathsf{PRK}_{3e2m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{3e2m}, {Y_s}^{x_e})$
$t_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$
if $t_2' \neq t_2$:
  $\mathtt{status} \leftarrow \mathsf{rejected}$
  return $\perp$
$\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$
$\mathsf{sk}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
$\mathsf{IV}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$
$\mathsf{salt}_{4e3m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
$\mathsf{PRK}_{4e3m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{4e3m}, {Y_e}^{x_s})$
$\mathtt{status} \leftarrow \mathsf{accepted}$
$\mathsf{CTX}_3 \leftarrow (\mathsf{ID_I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$
$t_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$
$m_3 \leftarrow (\mathsf{ID_I} \| t_3 \| \mathsf{EAD}_3)$
$c_3 \leftarrow \mathcal{E}'(\mathsf{sk}_3, \mathsf{IV}_3, m_3)$
$\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$
$\mathsf{PRK_{out}} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$
$\mathtt{status} \leftarrow \mathsf{terminated}$
$\mathtt{SK} \leftarrow \mathsf{PRK_{out}}$
return c_3

Run($\mathsf{ID}, \mathsf{sk}, \mathtt{peerpk}, m$)

if $\mathtt{status} \neq \mathsf{running}$:
  return $\perp$
if $\mathtt{role} = \mathsf{initiator}$:
  $m' \leftarrow \mathsf{InitRun2}(\mathsf{ID}, \mathsf{sk}, m)$
elseif $\mathsf{sid} = \perp$:
  $m' \leftarrow \mathsf{RespRun1}(\mathsf{ID}, \mathsf{sk}, m)$
else :
  $m' \leftarrow \mathsf{RespRun2}(\mathsf{ID}, \mathtt{peerpk}, m)$
return m

RespRun1($\mathsf{ID_R}, y_s, m_1 = (X_e, \mathsf{C_I}, \mathsf{EAD}_1)$)

$y_e \xleftarrow{\$} \mathbb{Z}_p$
$Y_e \leftarrow g^{y_e}$
$\mathsf{C_R} \xleftarrow{\$} \{0,1\}^{nl}$
$\mathsf{sid} \leftarrow (\mathsf{C_I}, \mathsf{C_R}, X_e, Y_e)$
$\mathsf{PRK}_{2e} \leftarrow \mathsf{RO}_T(\text{""}, {X_e}^{y_e})$
$\mathsf{sk}_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$
$\mathsf{salt}_{3e2m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
$\mathsf{PRK}_{3e2m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{3e2m}, {X_e}^{y_s})$
$\mathsf{TH}_2 \leftarrow \mathcal{H}(Y_e, \mathsf{C_R}, \mathcal{H}(m_1))$
$\mathsf{CTX}_2 \leftarrow (\mathsf{ID_R} \| \mathsf{TH}_2 \| Y_s \| \mathsf{EAD}_2)$
$t_2 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$
$m_2 \leftarrow (\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2)$
$c_2 \leftarrow \mathcal{E}(\mathsf{sk}_2, m_2)$
return $(Y_e, c_2, \mathsf{C_R})$

RespRun2($\mathsf{ID_R}, \mathtt{peerpk}, c_3$)

$\mathsf{TH}_3 \leftarrow \mathcal{H}(\mathsf{TH}_2, m_2)$
$\mathsf{sk}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
$\mathsf{IV}_3 \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$
$m_3 \leftarrow \mathcal{D}'(\mathsf{sk}_3, \mathsf{IV}_3, c_3)$
if $m_3 = \perp$:
  $\mathtt{status} \leftarrow \mathsf{rejected}$
  return $\perp$
$(\mathsf{ID_I} \| t_3 \| \mathsf{EAD}_3) \leftarrow m_3$
$X_s \leftarrow \mathtt{peerpk}[\mathsf{ID_I}]$
$\mathsf{salt}_{4e3m} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
$\mathsf{PRK}_{4e3m} \leftarrow \mathsf{RO}_T(\mathsf{salt}_{4e3m}, {X_s}^{y_e})$
$\mathtt{status} \leftarrow \mathsf{accepted}$
$\mathsf{CTX}_3 \leftarrow (\mathsf{ID_I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$
$t_3' \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$
if $t_3' \neq t_3$:
  $\mathtt{status} \leftarrow \mathsf{rejected}$
  return $\perp$
$\mathsf{TH}_4 \leftarrow \mathcal{H}(\mathsf{TH}_3, m_3)$
$\mathsf{PRK_{out}} \leftarrow \mathsf{RO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$
$\mathtt{status} \leftarrow \mathsf{terminated}$
$\mathtt{SK} \leftarrow \mathsf{PRK_{out}}$
return $\perp$

Fig. 5. Formalized description of the EDHOC protocol

respectively, and ℓ_{mac} for the tags. We thus define a simulation oracle by digest size: $\mathsf{SO}_P^{\mathsf{size}}$, for size in $\{\ell_2, \ell_{\mathsf{hash}}, \ell_{\mathsf{key}}, \ell_{\mathsf{iv}}, \ell_{\mathsf{mac}}\}$
The simulation oracles SO_P and $\mathsf{SO}_{\mathcal{H}}$ work as the usual way of simulating the answer with a new random answer for any new query, and the same answer if the same query is asked again. For the simulation oracles SO_T, the oracle consists in a list that contains elements of the form $(\mathsf{str}, Z, (X, Y); \lambda)$, where when first set, either Z or (X, Y) is non-empty. Indeed, when making a call to a random oracle, the official query is of the form (str, Z), where str is any bit string, that can be empty or a pseudo-random key, and Z is a Diffie-Hellman value. Then, the simulator checks in the list for an entry matching with $(\mathsf{str}, Z, *; \lambda)$. If such an element is found, one outputs λ, otherwise one randomly set $\lambda \xleftarrow{\$} \{0,1\}^\kappa$ and append $(\mathsf{str}, Z, \perp; \lambda)$ to the list. But later, the simulator will also ask queries of the form $(\mathsf{str}, (X, Y))$, where (X, Y) is a pair of group elements. Then one checks in the list for an entry matching with either $(\mathsf{str}, *, (X, Y); \lambda)$ or $(\mathsf{str}, Z, *; \lambda)$ such that $\mathsf{DDH}(g, X, Y, Z) = 1$. If such an element is found, one outputs λ, otherwise one randomly set $\lambda \xleftarrow{\$} \{0,1\}^\kappa$ and append $(\mathsf{str}, \perp, (X, Y); \lambda)$ to the list. When such new kinds of elements exist in the list, for the first kind of queries (str, Z), one checks in the list for an entry matching with either $(\mathsf{str}, Z, *; \lambda)$ as before, or $(\mathsf{str}, *, (X, Y); \lambda)$ such that $\mathsf{DDH}(g, X, Y, Z) = 1$. We detail in Figure 6 the functioning of those oracles, and the modifications made to the simulation.
Thanks to the DDH oracle, this simulation is perfect, and is thus indistinguishable to the adversary:

$$\Pr[\textsc{Succ}_1] = \Pr[\textsc{Succ}_0]$$

Game G_2. In order to prevent collisions in the future PRK generation, we modify the simulation oracles $\mathsf{SO}_T, \mathsf{SO}_P^{\ell_{\mathsf{hash}}}$ and $\mathsf{SO}_{\mathcal{H}}$, such that if a collision occurs, the simulator stops. We therefore need to determine the probability of a collision, to bound the probability for an adversary to distinguish this game from the previous one. To do so, we rely on the birthday paradox. By denoting $q_{\mathsf{SO}_T}, q_{\mathsf{SO}_P^{\ell_{\mathsf{hash}}}}, q_{\mathsf{SO}_{\mathcal{H}}}$ the amount of queries made to oracles $\mathsf{SO}_T, \mathsf{SO}_P^{\ell_{\mathsf{hash}}}, \mathsf{SO}_{\mathcal{H}}$ respectively, the birthday paradox bound gives:

$$\Pr[\textsc{Succ}_2] - \Pr[\textsc{Succ}_1] \leq \frac{{q_{\mathsf{SO}_T}}^2 + {q_{\mathsf{SO}_P^{\ell_{\mathsf{hash}}}}}^2 + {q_{\mathsf{SO}_{\mathcal{H}}}}^2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_3. One can note that thanks to the above simulation of the random oracles, the simulator does not need anymore to compute Diffie-Hellman values. Then, for every simulated player, the simulator generates X_e or Y_e at random in the group, and the simulation is still performed as in the previous game. As corruption queries only reveal long-term secret, still known to the simulator, the view of the adversary is perfectly indistinguishable of the previous game and we have:

$$\Pr[\textsc{Succ}_3] = \Pr[\textsc{Succ}_2]$$

$\mathsf{SO}_T(\mathsf{str}, \mathsf{input})$

1 : **if** $\mathsf{len}(\mathsf{input}) = 1$:
2 : $Z \leftarrow \mathsf{input}$
3 : **if** $\exists(\mathsf{str}, Z, *; \lambda) \in \mathsf{SO}_T$:
4 : **return** λ
5 : **else** :
6 : **if** $\exists(\mathsf{str}, \perp, (X, Y); \lambda) \in \mathsf{SO}_T$ s.t. $\mathsf{DDH}(X, Y, Z) = 1$:
// update SO_T
7 : ${\mathsf{SO}_T}^{-1}[\lambda] \leftarrow (\mathsf{str}, Z, (X, Y); \lambda)$
8 : **return** λ
9 : **else** :
10 : $\lambda \xleftarrow{\$} \{0,1\}^\kappa$
11 : $\mathsf{SO}_T \leftarrow \mathsf{SO}_T \cup \{(\mathsf{str}, Z, \perp; \lambda)\}$
12 : **return** λ
13 : **else** :
// $\mathsf{input} = (X, Y)$, only by the simulator
14 : $(X, Y) \leftarrow \mathsf{input}$
15 : **if** $\exists(\mathsf{str}, *, (X, Y); \lambda) \in \mathsf{SO}_T$:
16 : **return** λ
17 : **else** :
18 : **if** $\exists(\mathsf{str}, Z, \perp; \lambda) \in \mathsf{SO}_T$ s.t. $\mathsf{DDH}(X, Y, Z) = 1$:
19 : ${\mathsf{SO}_T}^{-1}[\lambda] \leftarrow (\mathsf{str}, Z, (X, Y); \lambda)$
20 : **return** λ
21 : **else** :
22 : $\lambda \xleftarrow{\$} \{0,1\}^\kappa$
23 : $\mathsf{SO}_T \leftarrow \mathsf{SO}_T \cup \{(\mathsf{str}, \perp, (X, Y); \lambda)\}$
24 : **return** λ

$\mathsf{RespRun1}(\mathsf{ID}_\mathbf{R}, y_s, m)$

5 : $\mathsf{PRK}_{2e} \leftarrow \mathsf{SO}_T(\text{""}, {X_e}^{y_e})$
6 : $\mathsf{sk}_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$
7 : $\mathsf{salt}_{3e2m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8 : $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{SO}_T(\mathsf{salt}_{3e2m}, {X_e}^{y_s})$
9 : $\mathsf{TH}_2 \leftarrow \mathsf{SO}_\mathcal{H}(Y_e, \mathsf{C_R}, \mathsf{SO}_\mathcal{H}(m_1))$
:
11 : $t_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$

$\mathsf{InitRun2}(\mathsf{ID}_\mathbf{I}, x_s, m)$

1 : $\mathsf{PRK}_{2e} \leftarrow \mathsf{SO}_T(\text{""}, {Y_e}^{x_e})$
2 : $\mathsf{TH}_2 \leftarrow \mathsf{SO}_\mathcal{H}(Y_e, \mathsf{C_R}, \mathsf{SO}_\mathcal{H}(m_1))$
3 : $\mathsf{sk}_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 0, \mathsf{TH}_2, \ell_2)$
:
7 : $\mathsf{salt}_{3e2m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{2e}, 1, \mathsf{TH}_2, \ell_{\mathsf{hash}})$
8 : $\mathsf{PRK}_{3e2m} \leftarrow \mathsf{SO}_T(\mathsf{salt}_{3e2m}, {Y_s}^{x_e})$
9 : $t_2 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 2, \mathsf{CTX}_2, \ell_{\mathsf{mac}})$
:
13 : $\mathsf{TH}_3 \leftarrow \mathsf{SO}_\mathcal{H}(\mathsf{TH}_2, m_2)$
14 : $\mathsf{sk}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
15 : $\mathsf{IV}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$
16 : $\mathsf{salt}_{4e3m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
17 : $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{SO}_T(\mathsf{salt}_{4e3m}, {Y_e}^{x_s})$
:
20 : $t_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$
:
23 : $\mathsf{TH}_4 \leftarrow \mathsf{SO}_\mathcal{H}(\mathsf{TH}_3, m_3)$
24 : $\mathsf{PRK}_{\mathbf{out}} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$

$\mathsf{RespRun2}(\mathsf{ID}_\mathbf{R}, \mathtt{peerpk}, c_3)$

1 : $\mathsf{TH}_3 \leftarrow \mathsf{SO}_\mathcal{H}(\mathsf{TH}_2, m_2)$
2 : $\mathsf{sk}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 3, \mathsf{TH}_3, \ell_{\mathsf{key}})$
3 : $\mathsf{IV}_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 4, \mathsf{TH}_3, \ell_{\mathsf{iv}})$
:
10 : $\mathsf{salt}_{4e3m} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{3e2m}, 5, \mathsf{TH}_3, \ell_{\mathsf{hash}})$
11 : $\mathsf{PRK}_{4e3m} \leftarrow \mathsf{SO}_T(\mathsf{salt}_{4e3m}, \mathtt{peerpk}[\mathsf{ID}_\mathbf{I}]^{y_e})$
:
14 : $t'_3 \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 6, \mathsf{CTX}_3, \ell_{\mathsf{mac}})$
:
18 : $\mathsf{TH}_4 \leftarrow \mathsf{SO}_\mathcal{H}(\mathsf{TH}_3, m_3)$
19 : $\mathsf{PRK}_{\mathbf{out}} \leftarrow \mathsf{SO}_P(\mathsf{PRK}_{4e3m}, 7, \mathsf{TH}_4, \ell_{\mathsf{hash}})$

Fig. 6. Description of SO_T list queries and modifications to the simulation

Game G_4. In this game, when simulating any **initiator** receiving a forged tuple $(Y_e, c_2, \mathsf{C_R})$ from the adversary in the name of a **non-corrupted user**, one simulates PRK_{3e2m} thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{3e2m}}$, which makes it perfectly unpredictable to the adversary. If the pair $(Y_e, \mathsf{C_R})$ is forged, TH_2 and salt_{3e2m} are different from the values obtained by a possibly simulated responder, thanks to the absence of collisions as they are respectively computed using $\mathsf{SO}_\mathcal{H}$ and $\mathsf{SO}_P^{\ell_{\mathsf{hash}}}$. Otherwise, sk_2 is not modified. So if the ciphertext c_2 is forged, thanks to the injective property of the encryption scheme $(\mathcal{E}, \mathcal{D})$ when the key is fixed, m_2, and by consequent TH_3 and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{3e2m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value ${X_e}^{y_s}$. We denote the event F_1, that query ${X_e}^{y_s}$ is asked whereas y_s is the long-term secret key of a non-corrupted user and X_e has been generated

by the simulator. If this event happens (which can easily be checked as the simulator knows y_s), one stops the simulation:

$$|\Pr[\text{Succ}_4] - \Pr[\text{Succ}_3]| \leq \Pr[F_1].$$

Game $G_{4'}$. We now provide an upper-bound on $\Pr[F_1]$: given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the X_e as $X_e = X \cdot g^r$, for random $r \xleftarrow{\$} \mathbb{Z}_p$, but chooses one user to set $Y_s = Y$. Even if y_s is therefore not known, simulation is still feasible as the simulator can make query to the SO_T oracle with input (X_e, Y_s). Then, one can still answer all the corruption queries, excepted for that user. But anyway, if F_1 happens on that user, this user must be non-corrupted at that time: one has solved the GDH problem, and one can stop the simulation. If the guess on the user is incorrect, one can also stop the simulation: $\Pr[F_1] \leq N \cdot \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}})$, where N is the number of users in the system.

Game G_5. In this game, when simulating any **responder** receiving a forged message m_1 from the adversary in the name of a **non-corrupted user**, still non-corrupted when sending c_3 to $\mathsf{RespRun2}$, one simulates PRK_{4e3m} thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{4e3m}}$, which makes it perfectly unpredictable to the adversary. Since m_1 is forged, thanks to the absence of collisions, TH_2,TH_3, and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{4e3m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value ${Y_e}^{x_s}$. We denote the event F_2, that query ${Y_e}^{x_s}$ is asked whereas x_s is the long-term secret key of a non-corrupted user and Y_e has been generated by the simulator. If this event happens (which can easily be checked as the simulator knows x_s), one stops the simulation:

$$|\Pr[\text{Succ}_5] - \Pr[\text{Succ}_4]| \leq \Pr[F_2].$$

Game $G_{5'}$. We now provide an upper-bound on $\Pr[F_2]$: given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the Y_e as $Y_e = Y \cdot g^{r'}$, for random $r' \xleftarrow{\$} \mathbb{Z}_p$, but chooses one user to set $X_s = X$. Then, one can still answer all the corruption queries, excepted for that user. But anyway, if F_2 happens on that user, this user must be non-corrupted at that time: one has solved the GDH problem, and one can stop the simulation. If the guess on the user is incorrect, one can also stop the simulation: $\Pr[F_2] \leq N \cdot \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}})$.

Game G_6. In this game, we simulate the key generation of PRK_{2e}, for all the passive sessions (m_1 received by a simulated responder comes from a simulated initiator, or $(Y_e, c_2, \mathsf{C_R})$ received by a simulated initiator comes from a simulated responder, and both used the same m_1 as first message), thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{2e}}$, acting in the same vein as SO_T, but not available to the adversary. This makes a difference with the previous game if the key PRK_{2e} has also been generated by asking SO_T on the correct Diffie-Hellman value $Z = g^{x_e y_e}$. We denote by F_3 the latter event, and stop the simulation in such a case:

$$|\Pr[\text{Succ}_6] - \Pr[\text{Succ}_5]| \leq \Pr[F_3]$$

Game $G_{6'}$. We now provide an upper-bound on $\Pr[F_3]$. Given a GDH challenge $(X = g^x, Y = g^y)$, one simulates all the X_e as $X_e = X \cdot g^r$, for random $r \xleftarrow{\$} \mathbb{Z}_p$, and all the Y_e as $Y_e = Y \cdot g^{r'}$, for random $r' \xleftarrow{\$} \mathbb{Z}_p$. As the key PRK_{2e} does not depend on the session context, any query Z to the SO_T oracle can make F_3 occurs on any of the passive session pairs $(X_e = X \cdot g^r, Y_e = Y \cdot g^{r'})$, we upper-bound the number by n_σ. Hence, q_{RO} DDH-oracle queries might be useful to detect F_3 on an input $Z = \mathsf{CDH}(X_e, Y_e) = g^{xy} \cdot X^{r'} \cdot Y^r \cdot g^{rr'}$, solving the GDH challenge (X, Y):

$$\Pr[F_3] \leq \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}).$$

Game G_7. In this game, when simulating any **initiator** receiving the second message $(Y_e, c_2, \mathsf{C_R})$, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{3e2m} thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{3e2m}}$. This makes a difference with the previous game only if this is a passive session, in which case PRK_{2e} is unpredictable, and thus different from the public one excepted with probability $2^{-\ell_{\mathsf{hash}}}$. As there are no collision, salt_{3e2m} is different from the value obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{3e2m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value $X_e^{y_s}$, which is not possible as event F_1 stops the simulation:

$$|\Pr[\textsc{Succ}_7] - \Pr[\textsc{Succ}_6]| \leq \frac{1}{2^{\ell_{\mathsf{hash}}}}.$$

Game G_8. In this game, when simulating any **initiator** receiving the second message $(Y_e, c_2, \mathsf{C_R})$, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{4e3m} thanks to a private oracle $\mathsf{SO}_{\mathsf{PRK}_{4e3m}}$. In this case, PRK_{3e2m} is unpredictable, as well as salt_{4e3m} and PRK_{4e3m}:

$$\Pr[\textsc{Succ}_8] = \Pr[\textsc{Succ}_7].$$

Game G_9. In this game, when simulating any **responder** receiving c_3, from the adversary in the name of a **non-corrupted user**, one simulates PRK_{4e3m} thanks to the private oracle $\mathsf{SO}_{\mathsf{PRK}_{4e3m}}$. This makes a difference with the previous game only if this is not a passive session, in which case PRK_{2e} is unpredictable, and thus different from the public one excepted with probability $2^{-\ell_{\mathsf{hash}}}$. As there are no collision, salt_{3e2m}, PRK_{3e2m}, and salt_{4e3m} are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of PRK_{4e3m} with respect to the public oracle answer, the adversary must have asked SO_T on the correct Diffie-Hellman value $Y_e^{x_s}$, which is not possible as event F_2 stops the simulation:

$$|\Pr[\textsc{Succ}_9] - \Pr[\textsc{Succ}_8]| \leq \frac{1}{2^{\ell_{\mathsf{hash}}}}.$$

Game G_{10}. In this game, for any fresh session, one simulates $\mathsf{PRK}_{\mathsf{out}}$ thanks to the private oracle $\mathsf{SO}_{\mathsf{PRK}_{\mathsf{out}}}$. A session being fresh means that no corruption

of the party or of the partner occurred before the time of acceptance: the initiator is not corrupted before receiving $(Y_e, c_2, \mathsf{C_R})$ and the responder is not corrupted before receiving c_3. By consequent, they are not corrupted before PRK_{4e3m} was computed. We have seen above that in those cases, the key PRK_{4e3m} is generated using the private oracle $\mathsf{SO}_{\mathsf{PRK}_{4e3m}}$: it is unpredictable. The use of the private oracle $\mathsf{SO}_{\mathsf{PRK_{out}}}$ can only be detected if the query PRK_{4e3m} is asked to SO_P:

$$|\Pr[\textsc{Succ}_{10}] - \Pr[\textsc{Succ}_9]| \leq \frac{q_{\mathsf{SO}_P^{\ell_{\mathsf{hash}}}}}{2^{\ell_{\mathsf{hash}}}}.$$

Globally, one can note that the gap between the initial and the last games is upper-bounded by

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) \\
&\qquad + \frac{{q_{\mathsf{SO}_T}}^2 + {q_{\mathsf{SO}_P^{\ell_{\mathsf{hash}}}}}^2 + {q_{\mathsf{SO}_{\mathcal{H}}}}^2}{2^{\ell_{\mathsf{hash}}+1}} + \frac{2 + q_{\mathsf{SO}_P^{\ell_{\mathsf{hash}}}}}{2^{\ell_{\mathsf{hash}}}} \\
&\leq \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}}
\end{aligned}$$

Eventually, for all the fresh sessions, in the real case ($b = 0$), the private oracle is used, and outputs a random key, while in the random case ($b = 1$), the session key is random too:

$$\Pr[\textsc{Succ}_{10}] = \frac{1}{2}.$$

This concludes the proof.

4 Explicit Authentication

Explicit authentication (or mutual authentication) aims to ensure each participant has the material to compute the final session key (accepts) when the partner terminates. In the EDHOC protocol, this means the responder (resp. the initiator) owns the private long-term key y_s (resp x_s) associated to the long-term public key Y_s (resp. X_s), and the private ephemeral keys, when the partner terminates.

To do so, the responder uses y_s in RespRun1 to compute PRK_{3e2m} used for the tag t_2 and the key sk_3. In the same way, the initiator uses x_s to compute PRK_{4e3m}, used for the tag t_3. Furthermore, they both have to use their ephemeral keys to compute PRK_{2e}, used for sk_2.

Responder Authentication. Consider a simulated **initiator** receiving a forged message $(Y_e, c_2, \mathsf{C_R})$ from the adversary in the name of a **non-corrupted user**. In such a case, consider the modifications made in the key privacy proof up to the game G_7. Hence, we have replaced the generation of PRK_{3e2m} with a private oracle. Then the advantage of the adversary in breaking the explicit

Finalize

1 : **return** :

$$\forall \pi_u^i \text{ s.t. } \begin{cases} \pi_u^i.\mathtt{status} = \mathsf{terminated} \\ \pi_u^i.t_{acc} < \mathsf{revltk}_{\pi_u^i.\mathtt{peerid}} \end{cases}, \exists \pi_v^j \text{ s.t. } \begin{cases} \pi_u^i.\mathtt{peerid} = v \\ \pi_v^j.\mathtt{peerid} = u \\ \pi_u^i.\mathsf{sid} = \pi_v^j.\mathsf{sid} \\ \pi_u^i.\mathtt{role} \neq \pi_v^j.\mathtt{role} \\ \pi_v^j.\mathtt{status} = \mathtt{accepted} \end{cases}$$

Fig. 7. Finalize Function for the Explicit Authentication Security Game

authentication of the responder in this game is bounded by $\frac{1}{2^{\ell_{\mathsf{mac}}}}$, added to the gap induced by the modifications made up to the game G_7. This leads to the following theorem:

Theorem 2. *The above EDHOC protocol satisfies the responder authentication under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\boldsymbol{Adv}^{auth-resp}_{EDHOC}(t; q_{RO}, n_\sigma, N)$ upper-bounded by*

$$\boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, n_\sigma \cdot q_{RO}) + 2N \cdot \boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, q_{RO}) + \frac{{q_{RO}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}} + \frac{1}{2^{\ell_{\mathsf{mac}}}}$$

Optimal Reduction. One cannot expect more after these three flows, as the adversary can play the role of the responder with known y_e. Without knowing y_s, it just gets stuck to compute PRK_{3e2m} and thus t_2. But it can guess it (with probability $2^{-\ell_{\mathsf{mac}}}$), breaking authentication. But it will not know $\mathtt{SK}$.

Initiator Authentication. We now consider any **responder** receiving a forged message c_3 from the adversary in the name of a **non-corrupted user**. As above, considering the modifications made in the key privacy proof up to the game G_8, we have replaced the generation of PRK_{4e3m} with a private oracle. Then the advantage of the adversary in breaking the explicit authentication of the initiator in this game is bounded by $\frac{1}{2^{\ell_{\mathsf{mac}}}}$, added to the gap induced by the modifications made up to the game G_7. This leads to the following theorem:

Theorem 3. *The above EDHOC protocol satisfies the initiator authentication under the Gap Diffie-Hellman problem in the Random Oracle model, and the injectivity of $(\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\boldsymbol{Adv}^{auth-init}_{EDHOC}(t; q_{RO}, n_\sigma, N)$ upper-bounded by*

$$\boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, n_\sigma \cdot q_{RO}) + 2N \cdot \boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, q_{RO}) + \frac{{q_{RO}}^2 + 4}{2^{\ell_{\mathsf{hash}}+1}} + \frac{1}{2^{\ell_{\mathsf{mac}}}}$$

Optimal Reduction. One cannot expect more after these three flows, as the adversary can play the role of the initiator with known x_e. Without knowing x_s, it just gets stuck to compute PRK_{4e3m} and thus t_3. But it can guess it (with probability $2^{-\ell_{\mathsf{mac}}}$) and encrypt it, as it knows sk_3, breaking authentication. But it will not know $\mathtt{SK}$.

5 Identity Protection

Let us now consider anonymity, with identity protection. More precisely, we want to show that the initiator's identity ($\mathsf{ID_I}$) is protected against active adversaries, while responder's identity ($\mathsf{ID_R}$) is protected only against passive adversaries.

The values $\mathsf{ID_I}$ and $\mathsf{ID_R}$ are the authentication credentials containing the public authentication keys of I and R, respectively.

Responder's Identity Protection. The value $\mathsf{ID_R}$ is used in the computation of CTX_2 itself used to compute t_2, which together with $\mathsf{ID_R}$ constitute the first part of $m_2 = (\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2)$ whose encryption is c_2 under sk_2. For the sake of clarity, we set $\mathsf{EAD}_2 = \mathtt{""}$ as EAD_2 is independent from the identity of the responder and has no cryptographic purpose. As a responder, the passive adversary can only earn information about $\mathsf{ID_R}$ using the ciphertext c_2. We thus define the responder identity protection experiment as follows:

$\exp^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}$

1 : $\mathsf{ID}_{\mathsf{R}_0}, \mathsf{ID}_{\mathsf{R}_1} \leftarrow \mathcal{A}(\mathtt{peerid})$
2 : $m_1 \leftarrow \mathcal{A}(\mathsf{InitRun1}(.))$
3 : $b \leftarrow \{0,1\}$
4 : $\mathsf{ID_R} \leftarrow \mathsf{ID}_{\mathsf{R}_b}$
5 : $y_s \leftarrow \mathsf{sk}_{\mathsf{ID_R}}$
6 : $(Y_e, c_2, \mathsf{C_R}) \leftarrow \mathsf{RespRun1}(\mathsf{ID_R}, y_s, m_1)$
7 : $b' \leftarrow \mathcal{A}(c_2)$
8 : **return** $b = b'$

We define the advantage $\mathbf{Adv}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}$ of the adversary in breaking the responder mutual authentication of EDHOC by:

$$\mathbf{Adv}^{\mathsf{ID-resp}-b}_{\mathsf{EDHOC}}(t) = |\Pr[\exp^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}} = 1] - \Pr[\exp^{\mathsf{ID-resp}-1}_{\mathsf{EDHOC}} = 1]|$$

Theorem 4. *The above EDHOC protocol protects Responder's Identity under the Gap Diffie-Hellman problem in the Random Oracle model, the injectivity and the semantic security of $\Pi = (\mathcal{E}, \mathcal{D})$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\boldsymbol{Adv}^{\mathsf{ID-resp}-b}_{EDHOC}(t; q_{RO}, n_\sigma, N)$ upper-bounded by*

$$\boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, n_\sigma \cdot q_{RO}) + 2N \cdot \boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, q_{RO}) + \boldsymbol{Adv}^{ind}_{\Pi}(t) + \frac{{q_{RO}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_0. This game is $\exp^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}}$. The simulated initiator follows the protocol, computes $c_2 = \mathcal{E}(\mathsf{sk}_2, (\mathsf{ID_R} \| t_2))$ and sends Y_e, c_2, C_R to the adversary:

$$\Pr[\textsc{Succ}_0] = \Pr[\exp^{\mathsf{ID-resp}-0}_{\mathsf{EDHOC}} = 1]$$

Game G_1. In this game, we applied the modification made from G_0 up to G_6 in the key privacy proof.

$$|\Pr[\textsc{Succ}_1] - \Pr[\textsc{Succ}_0]| \leq \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, n_\sigma \cdot q_{\mathsf{RO}}) + 2N \cdot \mathbf{Adv}^{\mathsf{GDH}}_{\mathbb{G}}(t, q_{\mathsf{RO}}) + \frac{{q_{\mathsf{RO}}}^2}{2^{\ell_{\mathsf{hash}}+1}}$$

Game G_2. In this game, one simulates sk_2 thanks to a private oracle, which makes a difference only if the random PRK_{2e} is asked to the public oracle:

$$|\Pr[\textsc{Succ}_2] - \Pr[\textsc{Succ}_1]| \leq \frac{1}{2^{\ell_{\mathsf{hash}}}}$$

Game G_3. In this game, we replace the line 4 of the experiment by $\mathsf{ID_R} \leftarrow \mathsf{ID}_{\mathsf{R}_{1-b}}$, leading to the instanciation of $\exp^{\mathsf{ID-resp}-1}_{\mathsf{EDHOC}}$. As sk_2 is chosen at random, using the semantic security of the encryption scheme $(\mathcal{E}, \mathcal{D})$, we thus have

$$|\Pr[\textsc{Succ}_3] - \Pr[\textsc{Succ}_2]| \leq \mathbf{Adv}^{\mathsf{ind}}_{\Pi}(t)$$

Initiator's Identity Protection. In this case, we expect an active security: we consider the simulation of an initiator interacting with an adversary playing in the name of a non-corrupted user with public long-term key Y_s. We have simulated PRK_{3e2m} with a private oracle, which leads to a private random key sk_3, unless the query has been asked, with the same argument as above.

The value $\mathsf{ID_I}$ is used in the computation of CTX_3 itself used to compute t_3, which together with $\mathsf{ID_I}$ constitute the first part of the message $m_3 = (\mathsf{ID_R} \| t_2 \| \mathsf{EAD}_2)$ whose encryption is c_3 under sk_3. As above, for the sake of generality, we set $\mathsf{EAD}_3 =$"". One can note that the first message m_1 sent by the initiator is independent of $\mathsf{ID_R}$. We therefore start the experiment after the adversary sent his first message $(Y_e, c_2, \mathsf{C_R})$:

$\exp^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}$

1 : $\mathsf{ID}_{\mathsf{I}_0}, \mathsf{ID}_{\mathsf{I}_1} \leftarrow \mathcal{A}(\texttt{peerid})$
2 : $(Y_e, c_2, \mathsf{C_R}) \leftarrow \mathcal{A}(\mathsf{RespRun1}(.))$
3 : $b \leftarrow \{0, 1\}$
4 : $\mathsf{ID_I} \leftarrow \mathsf{ID}_{\mathsf{I}_b}$
5 : $x_s \leftarrow \mathsf{sk}_{\mathsf{ID_I}}$
6 : $Y_s \leftarrow \texttt{peerpk}[\mathsf{ID_I}]$
7 : $c_3 \leftarrow \mathsf{InitRun2}(\mathsf{ID_I}, x_s, Y_s, (Y_e, c_2, \mathsf{C_R}))$
8 : $b' \leftarrow \mathcal{A}(c_3)$
9 : **return** $b = b'$

We define the advantage $\mathbf{Adv}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}$ of the adversary in breaking the responder mutual authentication of EDHOC by:

$$\mathbf{Adv}^{\mathsf{ID-init}-b}_{\mathsf{EDHOC}}(t) = |\Pr[\exp^{\mathsf{ID-init}-0}_{\mathsf{EDHOC}} = 1] - \Pr[\exp^{\mathsf{ID-init}-1}_{\mathsf{EDHOC}} = 1]|$$

Theorem 5. *The above EDHOC protocol protects Initiator's Identity under the Gap Diffie-Hellman problem in the Random Oracle model, the injectivity of $(\mathcal{E}, \mathcal{D})$ and the semantic security of $\Pi' = (\mathcal{E}', \mathcal{D}')$. More precisely, with q_{RO} representing the global number of queries to the random oracles, n_σ the number of running sessions, N the number of users, and ℓ_{hash} the hash digest length, we have $\boldsymbol{Adv}^{ID-init-b}_{EDHOC}(t; q_{RO}, n_\sigma, N)$ upper-bounded by*

$$\boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, n_\sigma \cdot q_{RO}) + 2N \cdot \boldsymbol{Adv}^{GDH}_{\mathbb{G}}(t, q_{RO}) + \boldsymbol{Adv}^{ind}_{\Pi'}(t) + \frac{{q_{RO}}^2 + 2}{2^{\ell_{\mathsf{hash}}+1}}$$

6 Improvements

We here make some remarks on the initial protocol, with some improvements that appear in gray highlights in Figure 8, and to the removed/additional hatched patterns in Figure 9.

6.1 On Mutual Authentication

The encryption key sk_3, used by the initiator to encrypt its second message m_3, is computed by calling HKDF-Expand on PRK_{3e2m}. However, even an adversary that plays in the name of a non-corrupted user, is able to compute PRK_{3e2m}, when knowing the Initiator ephemeral key x_e, as PRK_{3e2m} does not depend on x_s, the long term secret key of the Initiator. In order to break the Initiator authentication, with respect to a Responder, an adversary can play on behalf of any user as an Initiator. It will be able to compute sk_3, but not t_3, for which value it will need some luck, but this is only 64-bit long! Which is not enough for a 128-bit security.

To get around this issue, we suggest to modify the construction of Initiator's second message as follows: Initial message $m_3 = (\mathsf{ID_I} \| t_3 || \mathsf{EAD}_3)$ is split as $m_3 \leftarrow (\mathsf{ID_I})$ and $m'_3 \leftarrow (t_3 || \mathsf{EAD}_3)$. Thus, m_3 is encrypted using sk_3 (with a one-time pad encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$, under sk_3 still depending on PRK_{3e2m}) into c_3. Then m'_3 does not need to be encrypted. We introduce the value ℓ_{sec}, always set as the expected bit-security parameter, independently of the ℓ_{mac} value. Then, we set the length of t_3 to be ℓ_{sec}, as it already authenticates $\mathsf{CTX}_3 = (\mathsf{ID_I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3)$. Concretely, the second message sent by the initiator to the responder is:

$$c_3 \| m'_3, \text{ where } c_3 = \mathcal{E}(\mathsf{sk}_3, m_3), m'_3 = t_3 || \mathsf{EAD}_3$$

Once the Responder receives (c_3, m'_3), he first decrypts c_3, retrieves X_s using m_3, computes PRK_{4e3m} and is then able to verify the tag t_3, allowing to check the

authenticity of $\mathsf{ID_I}$, as well as all the other values is $\mathsf{CTX}_3 = \mathsf{ID_I} \| \mathsf{TH}_3 \| X_s \| \mathsf{EAD}_3$. The extra required length for the tag t_3 is perfectly compensated by the absence of the tag jointly sent when using Authenticated Encryption, and the plaintext length of m_3 is the same as the encryption of m_3. Therefore, this does not impact the communication cost of the protocol, until $\ell_{\mathsf{sec}} \leq 2 \times \ell_{\mathsf{mac}}$, but improves to ℓ_{sec}-bit security for Initiator-Authentication.

About the Responder-Authentication, t_2 also provides a 64-bit security level only: by guessing it, any active adversary can make the initiator terminate, and thus breaking the responder-authentication, if one does not wait for the fourth flow c_4, m'_4. However, with this fourth flow, we can show the $2 \times \ell_{\mathsf{mac}}$-bit security level is achieved.

6.2 On Reduction Efficiency

After analysis, we also notice another improvement: the key PRK_{2e} is computed according to $g^{x_e y_e}$ only, as the salt used in HKDF-Extract is an empty string. When considering several parellels sessions, this allows an adversary to find a collision with any of the session making a single call to HKDF-Extract. Therefore, we replace the empty string used as salt with TH_2 that depends on the session variables and is different for each session. Thus, an adversary has to make a call to HKDF-Extract with a chosen TH_2, linked to a specific session. This makes the reduction cost of the key-privacy game independent of n_σ, the number of sessions.

References

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

DG20. Hannah Davis and Felix Günther. Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. Cryptology ePrint Archive, Report 2020/1029, 2020. `https://eprint.iacr.org/2020/1029`.

NSB21. Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. Formal analysis of edhoc key establishment for constrained iot devices. In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT '21)*, pages 210–221. INSTICC, SciTePress, 2021. `https://arxiv.org/abs/2007.11427`.

Sch16. Jim Schaad. Cbor object signing and encryption (cose), 2016.

SMP22. Göran Selander, John Preuß Mattsson, and Francesca Palombini. Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-15, Internet Engineering Task Force, July 2022. `https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc/`.

Fig. 8. Optimized EDHOC with four messages in the STAT/STAT Authentication Method. Our modifications compared to [SMP22] (draft-ietf-lake-edhoc-15) are represented by $\boxed{\text{previous} \mid \text{new}}$ and additions by gray highlights

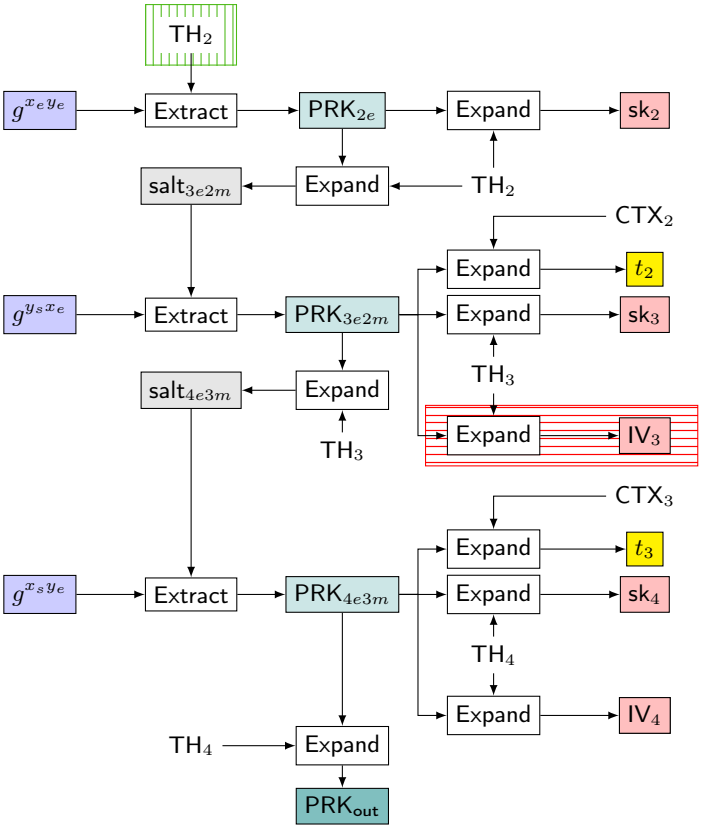

Fig. 9. Key Derivation (for the `STAT-STAT` Method) from [NSB21]. Green vertical hatchs denote additions and red horizontal hatchs denote removals compared to the initial version.