



**HAL**  
open science

# Security Analysis of Improved EDHOC Protocol

Baptiste Cottier, David Pointcheval

► **To cite this version:**

Baptiste Cottier, David Pointcheval. Security Analysis of Improved EDHOC Protocol. 15th International Symposium on Foundations & Practice of Security (FPS – 2022)., Dec 2022, Ottawa, Canada. hal-03772082v3

**HAL Id: hal-03772082**

**<https://hal.science/hal-03772082v3>**

Submitted on 23 Nov 2022

**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 Improved 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 SECURE '21, on a former version, leading to some improvements, in the ongoing evaluation process by IETF. Unfortunately, while formal analysis can detect some vulnerabilities in the protocol, it cannot evaluate the actual security level. In this paper, we study the protocol as it appeared in version 15. 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 Introduction

A key agreement is under analysis by IETF [SMP22], under the name *Ephemeral Diffie-Hellman Over COSE* (EDHOC). 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.

This protocol is deeply inspired from SIGMA [Kra03] and OPTLS [KW16] and targets constrained devices over low-power IoT radio communication technologies. For this reason, very aggressive parameters are proposed to minimize the communications. This paper follows a request from the LAKE working group to study the computational security of the EDHOC protocol with such aggressive parameters.

### 1.1 Related Work

A formal analysis of the May 2018 version has already been proposed by Bruni *et al.* in [BSJGPS18] and later completed and updated by [NSB21,CJJK22,JKKR23], leading to some improvements. But such a formal analysis, when successful, does not give any insight about the actual security level, in terms of time complexity of the best possible attack. While our computational analysis covers the MAC-based authentication method, other ongoing works cover other authentication methods based on signatures.

## 1.2 Contributions

In this paper, we analyse the August 2022 version of EDHOC proposal [SMP22]. We are able to prove the three expected security properties in the random oracle model, under a Diffie-Hellman assumption and with secure encryption primitives. However, because of the aggressive settings, we exhibit attacks in  $2^{64}$  operations, against authentication, which is not acceptable for a 128-bit security level.

We thereafter propose some improvements to get better security, at no communication cost. Firstly, adding more inputs to some hash value allows to speed-up the simulator when searching in some tables. Secondly, one converts an authenticated encryption scheme into a simple one-time secure encryption scheme, for hiding the identity of the Initiator, and sends a larger tag together with the External Authorization Data, in plaintext. We convert an authenticated ciphertext into a smaller ciphertext encrypting only a part of the message, and the remaining of the message is sent as plain values rather than encrypted, but with better authentication. This conversion globally has no communication impact, but increases from 64 to 128-bit security level for initiator-authentication. Last, we confirm that a fourth message provides a 128-bit security level for responder-authentication.

## 2 Preliminaries

### 2.1 Computational Assumptions

For security analysis in the computational setting, we rely on some computational assumptions: the Gap Diffie-Hellman problem and some properties of symmetric encryption.

*Gap Diffie-Hellman (GDH).* The Gap Diffie-Hellman problem aims to solve a Diffie-Hellman instance  $(U = g^u, V = g^v)$ , in a group  $\mathbb{G}$  with generator  $g$ , where  $u, v \xleftarrow{\$} \mathbb{Z}_p$ , by computing  $g^{uv}$ , with access to a Decisional Diffie-Hellman oracle DDH returning 1 if a tuple-query  $(g^a, g^b, g^c)$  is a Diffie-Hellman tuple, and 0 otherwise. We define the advantage  $\text{Adv}_{\mathbb{G}}^{\text{gdh}}(t, q_{\text{ddh}})$ , as the maximum advantage over all algorithms  $\mathcal{A}$  in outputting  $g^{uv}$ , with time-complexity at most  $t$  and making at most  $q_{\text{ddh}}$  queries to the DDH oracle.

*One-Time Pad Encryption.* We will use several symmetric encryption schemes, such as the one-time pad: given a random key  $\text{sk} \in \{0, 1\}^k$ , the encryption of the message  $m \in \{0, 1\}^k$  is  $c = \mathcal{E}(\text{sk}, m) \leftarrow m \oplus \text{sk}$ , while the decryption just consists in  $m = \mathcal{D}(\text{sk}, c) \leftarrow c \oplus \text{sk}$ . It satisfies the injective property:

$$\forall \text{sk}, m_0, m_1 \in \{0, 1\}^k, \mathcal{E}(\text{sk}, m_0) = \mathcal{E}(\text{sk}, m_1) \implies m_0 = m_1.$$

It also guarantees perfect privacy: for a random secret key  $\text{sk}$ ,  $c$  does not leak any information about the plaintext. We stress this is of course for a one-time use only, as there is no additional oracle access.

*Authenticated Encryption with Associated Data (AEAD)*. We will also use an Authenticated Encryption with Associated Data scheme  $\Pi' = (\mathcal{E}', \mathcal{D}')$ , with a key  $\text{sk}$  and initialisation vector  $\text{IV}$ . For a message  $m \in \mathcal{M}$  and some associated data  $a \in \mathcal{A}$ , the ciphertext is  $c = \mathcal{E}'(\text{sk}, \text{IV}; m; a)$ <sup>1</sup>, while the decryption process provides  $m = \mathcal{D}'(\text{sk}, \text{IV}; c; a)$  in case of valid ciphertext  $c$  with respect to  $\text{sk}$ ,  $\text{IV}$ , and  $a$ , or  $\perp$  otherwise. Two security properties are expected from such an AEAD.

**Indistinguishability.**  $\Pi' = (\mathcal{E}', \mathcal{D}')$  should protect message-privacy (IND-CPA, for Indistinguishability under Chosen-Plaintext Attacks). More precisely, we consider the Experiment  $\text{Exp}_{\Pi'}^{\text{ind-cpa}}(\mathcal{A})$  in which we randomly choose  $b \in \{0, 1\}$  and a secret key  $\text{sk}$ ,  $\mathcal{A}$  can ask multiple queries  $(\text{IV}, a, m_0, m_1)$ , all with different  $\text{IV}$ , and for each we compute and send  $c = \mathcal{E}'(\text{sk}, \text{IV}; m_b; a)$  to  $\mathcal{A}$ . Let  $b' \in \{0, 1\}$  be the output of  $\mathcal{A}$ . Then, the Experiment  $\text{Exp}_{\Pi'}^{\text{ind-cpa}}(\mathcal{A})$  outputs 1 if  $b' = b$  and 0 otherwise. We define the advantage of  $\mathcal{A}$  in violating IND-CPA security of  $\Pi'$  as  $\text{Adv}_{\Pi'}^{\text{ind-cpa}}(\mathcal{A}) = \Pr[\text{Exp}_{\Pi'}^{\text{ind-cpa}}(\mathcal{A}) = 1]$  and the advantage function  $\text{Adv}_{\Pi'}^{\text{ind-cpa}}(t)$ , as the maximum value of  $\text{Adv}_{\Pi'}^{\text{ind-cpa}}(\mathcal{A})$  over all  $\mathcal{A}$  with time-complexity at most  $t$ . We stress that  $c$  only aims at protecting the message-privacy, but does not provide any security for the associated data. Thanks to multiple queries, we are in the chosen-plaintext setting, and not a one-time security as before.

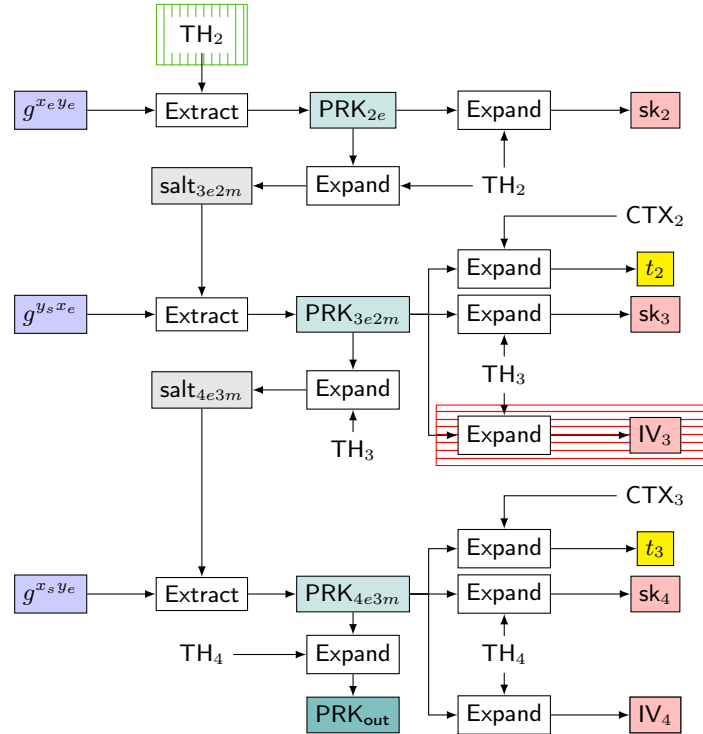
**Authentication.** An AEAD scheme is also expected to guarantee some unforgeability property (UF-CMA, for Unforgeability under Chosen-Message Attacks), also for the associated data (not encrypted). More precisely, we consider the Experiment  $\text{Exp}_{\Pi'}^{\text{uf-cma}}(\mathcal{A})$  in which  $\mathcal{A}$  is given access to the encryption oracle  $\mathcal{E}'(\text{sk}, \cdot; \cdot; \cdot)$ , for a random secret key  $\text{sk}$ . The Experiment returns 1 if  $\mathcal{A}$  outputs some data  $a$ , an initialisation vector  $\text{IV}$  and a ciphertext  $c$  accepted with respect to  $\text{IV}$  and  $a$ , which means that  $\mathcal{D}'(\text{sk}, \text{IV}; c; a) \neq \perp$ , while  $c$  has not been obtained as the output of an encryption query to  $\mathcal{E}'(\text{sk}, \cdot; \cdot; \cdot)$ . We define the advantage of  $\mathcal{A}$  in violating UF-CMA security of  $\Pi'$  as  $\text{Adv}_{\Pi'}^{\text{uf-cma}}(\mathcal{A}) = \Pr[\text{Exp}_{\Pi'}^{\text{uf-cma}}(\mathcal{A}) = 1]$  and the advantage function  $\text{Adv}_{\Pi'}^{\text{uf-cma}}(t)$  as the maximum value of  $\text{Adv}_{\Pi'}^{\text{uf-cma}}(\mathcal{A})$  over all  $\mathcal{A}$  with time-complexity at most  $t$ .

## 2.2 Brief Description of EDHOC

As with any key exchange protocol, EDHOC aims to provide a common session key to two parties. We briefly sketch the key elements of the EDHOC protocol. Due to the page limitations, we refer the reader to [SMP22] for a detailed description. 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).
- *Cipher Suites*: Ordered set of protocol security settings. Initial paper offers many possible suites, but we focus on the most aggressive cipher suites set-

<sup>1</sup> We use semicolons here to distinguish keying material, message and Additional Data.



**Fig. 1.** Key Derivation (for the STAT-STAT Method) from [NSB21]. Green vertical hatches denote additions and red horizontal hatches denote removals compared to the initial version.

ting the MAC length to 8 bytes, while still using SHA-256 as a hash function, with 256-bit outputs.

- *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 STAT/STAT. Also, we ignore 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, recalled in Figure 1 (ignoring the vertically hatched patterns for the initial protocol), the pseudo-random keys (PRK) are derived using an extraction function. In our context,  $\text{Extract}(\text{salt}, \text{IKM}) = \text{HKDF-Extract}(\text{salt}, \text{IKM})$  is defined with SHA-256, where IKM holds for Input Keying Material (in our context, this will be some Diffie-Hellman keys) and  $\text{Expand}(\text{PRK}, \text{info}, \text{len}) = \text{HKDF-Expand}(\text{PRK}, \text{info}, \text{len})$  where

$\mathbb{G} = \langle g \rangle$	Cyclic group generated by $g$ , of size $p$		
$\mathcal{H}$	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 Authorization Data		
$\mathcal{E}, \mathcal{D}$	One-time Pad Encryption and Decryption		
$\mathcal{E}', \mathcal{D}'$	AEAD Encryption and Decryption		
sk	Secret key	$t_2, t_3$	MAC tags
TH	Transcript Hash	$\perp$	Protocol abortion
$C_R, C_I$	Connection Identifiers	$\kappa_{\text{sec}}$	Expected bit-security

Lengths

$\ell_C$	Connection Identifiers	$\ell_{\text{mac}}, \ell_{\text{hash}}$	MAC and Hash output
$\ell_2, \ell_{\text{id}}$	$m_2$ and ID	$\ell_{\text{key}}, \ell_{\text{iv}}$	Key and IV

**Fig. 2.** Notations

info contains the transcript hash (TH<sub>2</sub>, TH<sub>3</sub> or TH<sub>4</sub>), the name of the derived key and some context, while len denotes the output length.

Transcript hashes, denoted TH<sub>*i*</sub>, are used as input to the HKDF-Expand function. More precisely, with SHA-256 as  $\mathcal{H}$ , we have:

$$\text{TH}_2 = \mathcal{H}(Y_e, C_R, \mathcal{H}(m_1)) \quad \text{TH}_3 = \mathcal{H}(\text{TH}_2, m_2) \quad \text{TH}_4 = \mathcal{H}(\text{TH}_3, m_3[, m'_3])$$

where  $m_1$  is the first message sent by the Initiator,  $m_2$  and  $m_3$  (possibly concatenated to  $m'_3$  in our improvement, to preserve the authentication property) respectively are the plaintexts respectively encrypted in the message 2 and message 3. More notations are provided in Figure 2.

*Protocol.* The detailed description of the initial protocol is given in Fig 3, ignoring the gray highlights which will be for our improvements. The final session key is  $\text{SK} = \text{PRK}_{\text{out}}$ .

### 3 Our Improvements

We here make some remarks on the initial protocol, with some improvements, that appear in gray highlights in Figure 3, and to the removed/additional hatched patterns in Figure 1.

#### 3.1 On Mutual Authentication

The encryption key  $\text{sk}_3$ , used by the initiator to encrypt its second message  $m_3$ , is computed by calling HKDF-Expand on  $\text{PRK}_{3e2m}$ . However, even an adversary that plays in the name of a non-corrupted user, is able to compute  $\text{PRK}_{3e2m}$ ,



**Fig. 3.** Optimized EDHOC with four messages in the STAT/STAT Authentication Method. Our modifications compared to [SMP22] (draft-ietf-lake-edhoc-15) are represented by previous | new and additions by gray highlights

when knowing the Initiator ephemeral key  $x_e$ , as  $\text{PRK}_{3e2m}$  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  $\text{sk}_3$ , 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  $m_3 = (\text{ID}_1 \| t_3 \| \text{EAD}_3)$  is split as  $m_3 \leftarrow (\text{ID}_1)$  and  $m'_3 \leftarrow (t_3 \| \text{EAD}_3)$ <sup>2</sup>. Thus,  $m_3$  is encrypted using  $\text{sk}_3$  (with a one-time pad encryption scheme  $\Pi = (\mathcal{E}, \mathcal{D})$ , under  $\text{sk}_3$  still depending on  $\text{PRK}_{3e2m}$ ) into  $c_3$ . Then  $m'_3$  does not need to be encrypted. We introduce the value  $\kappa_{\text{sec}}$ , always set as the expected bit-security parameter, independently of the  $\ell_{\text{mac}}$  value. Then, we set the length of  $t_3$  to be  $\kappa_{\text{sec}}$ , as it already authenticates  $\text{CTX}_3 = (\text{ID}_1 \| \text{TH}_3 \| X_s \| \text{EAD}_3)$ . Concretely, the second message sent by the initiator to the responder is:  $c_3 \| m'_3$ , where  $c_3 = \mathcal{E}(\text{sk}_3, m_3)$ ,  $m'_3 = t_3 \| \text{EAD}_3$ . Once the Responder receives  $(c_3, m'_3)$ , he first decrypts  $c_3$ , retrieves  $X_s$  using  $m_3$ , computes  $\text{PRK}_{4e3m}$  and is then able to verify the tag  $t_3$ , allowing to check the authenticity of  $\text{ID}_1$ , as well as all the other values is  $\text{CTX}_3 = \text{ID}_1 \| \text{TH}_3 \| X_s \| \text{EAD}_3$ . 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  $\kappa_{\text{sec}} \leq 2 \times \ell_{\text{mac}}$ , but improves to  $\kappa_{\text{sec}}$ -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  $2 \times \ell_{\text{mac}}$ -bit security level is achieved.

### 3.2 On Reduction Efficiency

After analysis, we also notice another improvement: the key  $\text{PRK}_{2e}$  is computed according to  $g^{x_e y_e}$  only, as the salt used in HKDF-Extract is an empty string. When considering several parallel 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  $\text{TH}_2$  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  $\text{TH}_2$ , linked to a specific session. This makes the reduction cost of the key-privacy game independent of the number of sessions.

## 4 Security Analysis

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

<sup>2</sup> One can move  $\text{EAD}_3$  in  $m_3$ , if privacy is required. It is still secure with any one-time secure encryption, but increasing the key size in the particular case of one-time pad.



- *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 distinguish 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. While the identity of the Initiator should be protected against active adversaries, the identity of the Responder should be protected against passive adversaries only.

*Random Oracle Model.* For the security analysis, we model Hash and Key Derivation Functions as random oracles. Respectively, the random oracles  $\text{RO}_T$  and  $\text{RO}_P$  will model HKDF-Extract and HKDF-Expand functions as perfect random functions.

#### 4.1 Key Privacy

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 *Sound* and *Fresh* predicates.

The advantage of an adversary  $\mathcal{A}$  against the key privacy is its bias in guessing  $b$ , from the random choice:  $\text{Adv}^{\text{kp-ake}}(\mathcal{A}) = \Pr[b' = b] - 1/2$ . For the reader's convenience, we give a formalized description of the EDHOC protocol in Figure 6, in the Appendix. It is 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. More precisely, with  $q_{RO}$  representing the global number of queries to the random oracles,  $N$  the number of users, and  $\ell_{\text{hash}}$  the hash digest length,  $\text{Adv}_{\text{EDHOC}}^{\text{kp-ake}}(t; q_{RO}, N)$  is upper-bounded by*

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

<b>Initialize ()</b> 1 : $\text{time} \leftarrow 0$ 2 : $\text{users} \leftarrow 0$ 3 : $b \leftarrow_{\$} \{0, 1\}$	<b>LongTermKeyReveal(<math>u</math>)</b> 1 : $\text{time} \leftarrow \text{time} + 1$ 2 : $\text{revltk}_u \leftarrow \text{time}$ 3 : <b>return</b> $\text{sk}_u$	<b>Finalize(<math>b'</math>)</b> 1 : <b>if</b> $\neg \text{Sound}$ : 2 : <b>return</b> 1 3 : <b>if</b> $\neg \text{Fresh}$ : 4 : $b' \leftarrow 0$ 5 : <b>return</b> $[b = b']$
<b>NewUser ()</b> 1 : $\text{users} \leftarrow \text{users} + 1$ 2 : $(\text{pk}_{\text{users}}, \text{sk}_{\text{users}}) \leftarrow_{\$} \text{KGen}$ 3 : $\text{revltk}_{\text{users}} \leftarrow \infty$ 4 : $\text{peerpk}[\text{users}] \leftarrow \text{pk}_{\text{users}}$ 5 : <b>return</b> $\text{pk}_{\text{users}}$	<b>SessionKeyReveal(<math>u, i</math>)</b> 1 : <b>if</b> $\pi_u^i = \perp$ or $\pi_u^i.\text{status} \neq \text{accepted}$ : 2 : <b>return</b> $\perp$ 3 : $\pi_u^i.\text{revealed} \leftarrow \text{true}$ 4 : <b>return</b> $\pi_u^i.\text{SK}$	
<b>Send(<math>u, i, m</math>)</b> 1 : <b>if</b> $\pi_u^i = \perp$ : 2 : $(\text{peerid}, \text{role}) \leftarrow m$ 3 : $(\pi_u^i, m') \leftarrow_{\$} \text{Activate}(u, \text{sk}_u, \text{peerid}, \text{peerpk}, \text{role})$ 4 : $\pi_u^i.t_{acc} \leftarrow 0$ 5 : <b>else</b> : 6 : $(\pi_u^i, m') \leftarrow_{\$} \text{Run}(u, \text{sk}_u, \pi_u^i, \text{peerpk}, \text{role})$ 7 : <b>if</b> $\pi_u^i.\text{status} = \text{accepted}$ : 8 : $\text{time} \leftarrow \text{time} + 1$ 9 : $\pi_u^i.t_{acc} \leftarrow \text{time}$ 10 : <b>return</b> $m'$	<b>Test(<math>u, i</math>)</b> 1 : <b>if</b> $\pi_u^i = \perp$ or $\pi_u^i.\text{status} \neq \text{accepted}$ or $\pi_u^i.\text{tested}$ 2 : <b>return</b> $\perp$ 3 : $\pi_u^i.\text{tested} \leftarrow \text{true}$ 4 : $T \leftarrow T \cup \{\pi_u^i\}$ 5 : $k_0 \leftarrow \pi_u^i.\text{SK}$ 6 : $k_1 \leftarrow_{\$} \text{KE.KS}$ 7 : <b>return</b> $k_b$	
<b>Sound</b> 1 : <b>if</b> $\exists$ distinct $\pi_u^i, \pi_v^j, \pi_w^k$ with $\pi_u^i.\text{sid} = \pi_v^j.\text{sid} = \pi_w^k.\text{sid}$ : 2 : <b>return</b> <b>false</b> 3 : <b>if</b> $\exists \pi_u^i, \pi_v^j$ with 4 : $\pi_u^i.\text{status} = \pi_v^j.\text{status} = \text{accepted}$ 5 :     and $\pi_u^i.\text{sid} = \pi_v^j.\text{sid}$ 6 :     and $\pi_u^i.\text{peerid} = u$ and $\pi_v^j.\text{peerid} = v$ 7 :     and $\pi_u^i.\text{role} \neq \pi_v^j.\text{role}$ , but $\pi_u^i.\text{SK} \neq \pi_v^j.\text{SK}$ 8 : <b>return</b> <b>false</b> 9 : <b>return</b> <b>true</b>	<b>Fresh</b> 1 : $\forall \pi_u^i \in T$ 2 : <b>if</b> $\pi_u^i.\text{revealed}$ or $\text{revltk}_{\pi_u^i.\text{peerid}} < \pi_u^i.t_{acc}$ : 3 : <b>return</b> <b>false</b> 4 : <b>if</b> $\exists \pi_v^j \neq \pi_u^i$ s.t. $\pi_u^i.\text{sid} = \pi_v^j.\text{sid}$ and $(\pi_v^j.\text{tested} \text{ or } \pi_v^j.\text{revealed})$ : 5 : <b>return</b> <b>false</b> 6 : <b>return</b> <b>true</b>	

**Fig. 4.** Authenticated Key Exchange Key Privacy Security Game  $G_{\text{AKE}, \mathcal{A}}^{\text{kp-ake}}$

**Game  $G_0$ .** This game is the key privacy security game  $G_{\text{AKE}, \mathcal{A}}^{\text{kp-ake}}$  (defined in Figure 4) played by  $\mathcal{A}$  using the KeyGen, Activate and Run algorithms (defined in Figure 6). 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  $\pi_u^i$ .

$$\Pr[\text{SUCC}_0] = \Pr[G_{\text{AKE}, \mathcal{A}}^{\text{kp-ake}}],$$

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 unre-

vealed. 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  $\text{RO}_T$  and  $\mathcal{H}$  always return a digest of size  $\ell_{\text{hash}}$ , we simply use the simulation oracle  $\text{SO}_T$  and  $\text{SO}_{\mathcal{H}}$  respectively. However,  $\text{RO}_P$  may return values of several lengths. We thus define a simulation oracle by digest size:  $\text{SO}_P^{\text{size}}$ , for size in  $\{\ell_2, \ell_{\text{id}}, \ell_{\text{hash}}, \ell_{\text{key}}, \ell_{\text{iv}}, \ell_{\text{mac}}, \kappa_{\text{sec}}\}$ . The simulation oracles  $\text{SO}_P$  and  $\text{SO}_{\mathcal{H}}$  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  $\text{SO}_T$ , the oracle consists in a list that contains elements of the form  $(\text{str}, Z, (X, Y); \lambda)$ , 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  $(\text{str}, Z)$ , where  $\text{str}$  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  $(\text{str}, Z, *; \lambda)$ . If such an element is found, one outputs  $\lambda$ , otherwise one randomly set  $\lambda \xleftarrow{\$} \{0, 1\}^\kappa$  and append  $(\text{str}, Z, \perp; \lambda)$  to the list. But later, the simulator will also ask queries of the form  $(\text{str}, (X, Y))$ , where  $(X, Y)$  is a pair of group elements. Then one checks in the list for an entry matching with either  $(\text{str}, *, (X, Y); \lambda)$  or  $(\text{str}, Z, *; \lambda)$  such that  $\text{DDH}(g, X, Y, Z) = 1$ . If such an element is found, one outputs  $\lambda$ , otherwise one randomly set  $\lambda \xleftarrow{\$} \{0, 1\}^\kappa$  and append  $(\text{str}, \perp, (X, Y); \lambda)$  to the list. When such new kinds of elements exist in the list, for the first kind of queries  $(\text{str}, Z)$ , one checks in the list for an entry matching with either  $(\text{str}, Z, *; \lambda)$  as before, or  $(\text{str}, *, (X, Y); \lambda)$  such that  $\text{DDH}(g, X, Y, Z) = 1$ . Thanks to the DDH oracle, this simulation is perfect, and is thus indistinguishable to the adversary:  $\Pr[\text{SUCC}_1] = \Pr[\text{SUCC}_0]$ .

**Game  $G_2$ .** In order to prevent collisions in the future PRK generation, we modify the simulation oracles  $\text{SO}_T$ ,  $\text{SO}_P^{\ell_{\text{hash}}}$  and  $\text{SO}_{\mathcal{H}}$ , such that if a collision occurs, the simulator stops. From the birthday paradox bound, we have:

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

**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:  $\Pr[\text{SUCC}_3] = \Pr[\text{SUCC}_2]$ .

**Game  $G_4$ .** In this game, when simulating any **initiator** receiving a forged tuple  $(Y_e, c_2, C_R)$  from the adversary in the name of a **non-corrupted user**, one simulates  $\text{PRK}_{3e2m}$  thanks to a private oracle  $\text{SO}_{\text{PRK}_{3e2m}}$ , which makes it perfectly unpredictable to the adversary. If the pair  $(Y_e, C_R)$  is forged,  $\text{TH}_2$

and  $\text{salt}_{3e2m}$  are different from the values obtained by a possibly simulated responder, thanks to the absence of collisions as they are respectively computed using  $\text{SO}_{\mathcal{H}}$  and  $\text{SO}_P^{\ell_{\text{hash}}}$ . Otherwise,  $\text{sk}_2$  is not modified. So if the ciphertext  $c_2$  is forged, thanks to the injective property of the one-time pad encryption scheme  $(\mathcal{E}, \mathcal{D})$  when the key is fixed,  $m_2$ , then  $\text{TH}_3$  and  $\text{salt}_{4e3m}$  are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of  $\text{PRK}_{3e2m}$  with respect to the public oracle answer, the adversary must have asked  $\text{SO}_T$  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  $\text{SO}_T$  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:  $\Pr[F_1] \leq N \cdot \text{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{\text{RO}})$ , 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  $\text{RespRun2}$ , one simulates  $\text{PRK}_{4e3m}$  thanks to a private oracle  $\text{SO}_{\text{PRK}_{4e3m}}$ , which makes it perfectly unpredictable to the adversary. Since  $m_1$  is forged, thanks to the absence of collisions,  $\text{TH}_2, \text{TH}_3$ , and  $\text{salt}_{4e3m}$  are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of  $\text{PRK}_{4e3m}$  with respect to the public oracle answer, the adversary must have asked  $\text{SO}_T$  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, as above, 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:  $\Pr[F_2] \leq N \cdot \text{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{\text{RO}})$ .

**Game  $G_6$ .** In this game, we simulate the key generation of  $\text{PRK}_{2e}$ , for all the passive sessions ( $m_1$  received by a simulated responder comes from a simulated initiator, or  $(Y_e, c_2, \text{C}_R)$  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  $\text{SO}_{\text{PRK}_{2e}}$ , acting in the same vein as  $\text{SO}_T$ , but not available

to the adversary. This makes a difference with the previous game if the key  $\text{PRK}_{2e}$  has also been generated by asking  $\text{SO}_T$  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  $\text{PRK}_{2e}$  now depends on the session context, any query  $Z$  to the  $\text{SO}_T$  oracle can make  $F_3$  occurs on a single pair  $(X_e = X \cdot g^r, Y_e = Y \cdot g^{r'})$ . Hence,  $q_{\text{RO}}$  DDH-oracle queries might be useful to detect  $F_3$  on an input  $Z = \text{CDH}(X_e, Y_e) = g^{xy} \cdot X^{r'} \cdot Y^r \cdot g^{rr'}$ , solving the GDH challenge  $(X, Y)$ :  $\Pr[F_3] \leq \text{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{\text{RO}})$ .

**Game  $G_7$ .** In this game, when simulating any **initiator** receiving the second message  $(Y_e, c_2, \text{C}_R)$ , from the adversary in the name of a **non-corrupted user**, one simulates  $\text{PRK}_{3e2m}$  thanks to a private oracle  $\text{SO}_{\text{PRK}_{3e2m}}$ . This makes a difference with the previous game only if this is a passive session, in which case  $\text{PRK}_{2e}$  is unpredictable, and thus different from the public one excepted with probability  $2^{-\ell_{\text{hash}}}$ . As there are no collisions,  $\text{salt}_{3e2m}$  is different from the value obtained by a possibly simulated responder. In order to detect the inconsistency of  $\text{PRK}_{3e2m}$  with respect to the public oracle answer, the adversary must have asked  $\text{SO}_T$  on the correct Diffie-Hellman value  $X_e^{y_e}$ , which is not possible as event  $F_1$  already stops the simulation. Hence, we just have  $|\Pr[\text{SUCC}_7] - \Pr[\text{SUCC}_6]| \leq 2^{-\ell_{\text{hash}}}$ .

**Game  $G_8$ .** In this game, when simulating any **initiator** receiving the second message  $(Y_e, c_2, \text{C}_R)$ , from the adversary in the name of a **non-corrupted user**, one simulates  $\text{PRK}_{4e3m}$  thanks to a private oracle  $\text{SO}_{\text{PRK}_{4e3m}}$ . In this case,  $\text{PRK}_{3e2m}$  is unpredictable, as well as  $\text{salt}_{4e3m}$  and  $\text{PRK}_{4e3m}$ :  $\Pr[\text{SUCC}_8] = \Pr[\text{SUCC}_7]$ .

**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  $\text{PRK}_{4e3m}$  thanks to the private oracle  $\text{SO}_{\text{PRK}_{4e3m}}$ . This makes a difference with the previous game only if this is not a passive session, in which case  $\text{PRK}_{2e}$  is unpredictable, and thus different from the public one excepted with probability  $2^{-\ell_{\text{hash}}}$ . As there are no collisions,  $\text{salt}_{3e2m}$ ,  $\text{PRK}_{3e2m}$ , and  $\text{salt}_{4e3m}$  are different from the values obtained by a possibly simulated responder. In order to detect the inconsistency of  $\text{PRK}_{4e3m}$  with respect to the public oracle answer, the adversary must have asked  $\text{SO}_T$  on the correct Diffie-Hellman value  $Y_e^{x_e}$ , which is not possible as event  $F_2$  already stops the simulation:  $|\Pr[\text{SUCC}_9] - \Pr[\text{SUCC}_8]| \leq 2^{-\ell_{\text{hash}}}$ .

**Game  $G_{10}$ .** In this game, for any fresh session, one simulates  $\text{PRK}_{\text{out}}$  thanks to the private oracle  $\text{SO}_{\text{PRK}_{\text{out}}}$ . 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  $(Y_e, c_2, \text{C}_R)$  and the responder is not corrupted before receiving  $c_3$ . By consequent, they are not corrupted before  $\text{PRK}_{4e3m}$  was computed. We have seen above that in those cases, the key  $\text{PRK}_{4e3m}$  is generated using the private oracle  $\text{SO}_{\text{PRK}_{4e3m}}$ : it is unpredictable.

The use of the private oracle  $\text{SO}_{\text{PRK}_{\text{out}}}$  can only be detected if the query  $\text{PRK}_{4e3m}$  is asked to  $\text{SO}_P$ :  $|\Pr[\text{SUCC}_{10}] - \Pr[\text{SUCC}_9]| \leq q_{\text{SO}_P^{\ell_{\text{hash}}}} \times 2^{-\ell_{\text{hash}}}$ . Globally, one can note that the gap between the initial and the last games is upper-bounded by

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

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:  $\Pr[\text{SUCC}_{10}] = 1/2$ . This concludes the proof.

## 4.2 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.

Finalize

---

1: **return** :

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

**Fig. 5.** Finalize Function for the Explicit Authentication Security Game

To do so, the responder uses  $y_s$  in  $\text{RespRun1}$  to compute  $\text{PRK}_{3e2m}$  used for the tag  $t_2$  and the key  $\text{sk}_3$ . In the same way, the initiator uses  $x_s$  to compute  $\text{PRK}_{4e3m}$ , used for the tag  $t_3$ . Furthermore, they both have to use their ephemeral keys to compute  $\text{PRK}_{2e}$ , used for  $\text{sk}_2$ .

*Responder Authentication.* Consider a simulated **initiator** receiving a forged message  $(Y_e, c_2, C_R)$  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  $\text{PRK}_{3e2m}$  with a private oracle. Then the advantage of the adversary in breaking the explicit authentication of the responder in this game is bounded by  $2^{-\ell_{\text{mac}}}$ , 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 property under the Gap Diffie-Hellman problem in the Random Oracle model. More precisely, with  $q_{RO}$  representing the global number of queries to the random oracles,  $N$  the number of users,  $\ell_{\text{hash}}$  the hash digest length and  $\ell_{\text{mac}}$  the MAC digest length, we have  $\mathbf{Adv}_{\text{EDHOC}}^{\text{auth-resp}}(t; q_{RO}, N)$  is upper-bounded by*

$$(2N + 1) \cdot \mathbf{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{RO}) + \frac{q_{RO}^2 + 2}{2^{\ell_{\text{hash}} + 1}} + \frac{1}{2^{\ell_{\text{mac}}}}.$$

**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  $\text{PRK}_{3e2m}$  and thus  $t_2$ . But it can guess it (with probability  $2^{-\ell_{\text{mac}}}$ ), breaking authentication. But it will not know  $\text{SK}$ . However, by waiting for the fourth message containing an authenticated encryption  $c_4$ , as said in the documentation, this will add a factor  $\mathbf{Adv}_{\Pi'}^{\text{uf-cma}}(t) \approx 2^{-\ell_{\text{mac}}}$  to the Responder Authentication security:  $\mathbf{Adv}_{\text{EDHOC}}^{\text{auth-resp}}(t; q_{RO}, N)$  is upper-bounded by

$$(2N + 1) \cdot \mathbf{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{RO}) + \frac{q_{RO}^2 + 2}{2^{\ell_{\text{hash}} + 1}} + \frac{1}{2^{\ell_{\text{mac}}}} \times \mathbf{Adv}_{\Pi'}^{\text{uf-cma}}(t).$$

*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  $\text{PRK}_{4e3m}$  with a private oracle. Then the advantage of the adversary in breaking the explicit authentication of the initiator in this game is bounded by  $\frac{1}{2^{\kappa_{\text{sec}}}}$ . 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 property under the Gap Diffie-Hellman problem in the Random Oracle model. More precisely, with  $q_{RO}$  representing the global number of queries to the random oracles,  $N$  the number of users,  $\ell_{\text{hash}}$  the hash digest length and  $\kappa_{\text{sec}}$  the expected bit-security, we have  $\mathbf{Adv}_{\text{EDHOC}}^{\text{auth-init}}(t; q_{RO}, N)$  upper-bounded by*

$$(2N + 1) \cdot \mathbf{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{RO}) + \frac{q_{RO}^2 + 4}{2^{\ell_{\text{hash}} + 1}} + \frac{1}{2^{\kappa_{\text{sec}}}}.$$

### 4.3 Identity Protection

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

The values  $\text{ID}_I$  and  $\text{ID}_R$  are the authentication credentials containing the public authentication keys of the Initiator and the Responder, respectively.

Both those values are sent to the other respective party using One-Time Pad encryption, that perfectly protects the privacy. Then, in one hand we have  $\text{ID}_R$  that is part of  $\text{CTX}_2$  used to compute  $t_2$  and in the other hand, we have  $\text{ID}_I$  that is part of  $\text{CTX}_3$  used to compute  $t_3$ . We thus define the similar responder and initiator identity protection experiment as follows:

$\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-resp-}b}$	$\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-init-}b}$
1 : $\text{ID}_{R_0}, \text{ID}_{R_1} \leftarrow \mathcal{A}(\text{peerid})$ 2 : $m_1 \leftarrow \mathcal{A}(\text{InitRun1}(\cdot))$ 3 : $b \leftarrow \{0, 1\}$ 4 : $\text{ID}_R \leftarrow \text{ID}_{R_b}$ 5 : $y_s \leftarrow \text{sk}_{\text{ID}_R}$ 6 : $(Y_e, c_2, C_R) \leftarrow \text{RespRun1}(\text{ID}_R, y_s, m_1)$ 7 : $b' \leftarrow \mathcal{A}(c_2)$ 8 : <b>return</b> $b = b'$	1 : $\text{ID}_{I_0}, \text{ID}_{I_1} \leftarrow \mathcal{A}(\text{peerid})$ 2 : $(Y_e, c_2, C_R) \leftarrow \mathcal{A}(\text{RespRun1}(\cdot))$ 3 : $b \leftarrow \{0, 1\}$ 4 : $\text{ID}_I \leftarrow \text{ID}_{I_b}$ 5 : $x_s \leftarrow \text{sk}_{\text{ID}_I}$ 6 : $Y_s \leftarrow \text{peerpk}[\text{ID}_I]$ 7 : $c_3 \leftarrow \text{InitRun2}(\text{ID}_I, x_s, Y_s, (Y_e, c_2, C_R))$ 8 : $b' \leftarrow \mathcal{A}(c_3)$ 9 : <b>return</b> $b = b'$

In both cases, we consider the modifications made in the key privacy proof up to the game  $G_7$ , making  $\text{PRK}_{2e}$  and  $\text{PRK}_{3e4m}$  random, and by consequent, so are  $\text{sk}_2$  and  $\text{sk}_3$ .

*Responder Identity Protection* The responder's identity has to be protected against passive adversaries only. To distinguish  $\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-resp-}0}$  and  $\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-resp-}1}$ , one must distinguish between an encryption of  $\text{ID}_{R_0}$  and  $\text{ID}_{R_1}$ , as  $\text{sk}_2$  is random, this implies breaking the injective property and the indistinguishability of  $\Pi = (\mathcal{E}, \mathcal{D})$ , both being perfect with the one-time pad.

*Initiator Identity Protection* The initiator's identity has to be protected against active adversaries. However, if the adversary plays in the name of a responder, he will be detected with high probability with the tag  $t_2$  before reaching game  $G_7$ . Therefore, distinguish between  $\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-init-}0}$  and  $\mathbf{Exp}_{\text{EDHOC}}^{\text{ID-init-}1}$  also implies breaking the injective property and the indistinguishability of  $\Pi = (\mathcal{E}, \mathcal{D})$ , both being perfect with the one-time pad.

**Theorem 4.** *The above EDHOC protocol protects Initiator and Responder's Identity under the Gap Diffie-Hellman problem in the Random Oracle model. More precisely, with  $q_{RO}$  representing the global number of queries to the random oracles,  $N$  the number of users, and  $\ell_{\text{hash}}$  the hash digest length, both advantages  $\mathbf{Adv}_{\text{EDHOC}}^{\text{ID-init-}b}(t; q_{RO}, N)$  and  $\mathbf{Adv}_{\text{EDHOC}}^{\text{ID-resp-}b}(t; q_{RO}, N)$  are upper-bounded by*

$$(2N + 1) \cdot \mathbf{Adv}_{\mathbb{G}}^{\text{GDH}}(t, q_{RO}) + \frac{q_{RO}^2 + 2}{2^{\ell_{\text{hash}} + 1}}.$$

## 5 Conclusion

Our computational analysis proved the EDHOC protocol instantiated with the STAT-STAT authentication method, with  $\ell_{\text{mac}} = 64$  and  $\kappa_{\text{sec}} = 128$ , provides nearly a 128-bit security level for key privacy and identity protection for both the responder and the initiator. In a three-flow scenario, Initiator Authentication reaches a 128-bit security level, using our improvements without extra-cost in our settings, but only a 64-bit security level for the responder. However, as suggested



in their documentation, a fourth message using authenticated encryption (AEAD) from the responder to the initiator increases this security up to a 128-bit level. Hence, our improvement of EDHOC, at no communication cost, provides a global 128-bit security level.

## Acknowledgments

This work was supported in part by the French ANR Project Crypto4Graph-AI.

## 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.
- BSJGPS18. Alessandro Bruni, Thorvald Sahl Jørgensen, Theis Grønbech Petersen, and Carsten Schürmann. Formal verification of ephemeral diffie-hellman over cose (edhoc). In Cas Cremers and Anja Lehmann, editors, *Security Standardisation Research*, pages 21–36, Cham, 2018. Springer International Publishing.
- CJKK22. Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. Sapic+: protocol verifiers of the world, unite! In *USENIX Security Symposium (USENIX Security), 2022, 2022*.
- 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>.
- JKKR23. Charlie Jacomme, Elise Klein, Steve Kremer, and Maiwenn Racouchot. A comprehensive, formal and automated analysis of the edhoc protocol. In *USENIX Security Symposium (USENIX Security), 2023 - To appear, 2023*.
- Kra03. Hugo Krawczyk. SIGMA: The “SIGn-and-MAC” approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, Heidelberg, August 2003.
- KW16. Hugo Krawczyk and Hoeteck Wee. The optls protocol and tls 1.3. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P '16)*, pages 81–96. IEEE Computer Society, 2016. <https://eprint.iacr.org/2015/978>.
- 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>.
- 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/>.

## A Formalized definition of the EDHOC protocol

<pre> KeyGen() ----- 1 : <math>sk \xleftarrow{\\$} \mathbb{Z}_p</math> 2 : <math>pk \leftarrow g^{sk}</math> 3 : <b>return</b> (pk, sk) Activate(ID, role) ----- 1 : <math>role \leftarrow role</math> 2 : <math>status \leftarrow running</math> 3 : <b>if</b> <math>role = initiator</math> : 4 :   <math>m' \leftarrow InitRun1(ID)</math> 5 : <b>else</b> <math>m' \leftarrow \perp</math> 6 : <b>return</b> <math>m</math> InitRun1(ID<sub>1</sub>) ----- 1 : <math>x_e \xleftarrow{\\$} \mathbb{Z}_p</math> 2 : <math>X_e \leftarrow g^{x_e}</math> 3 : <math>C_1 \xleftarrow{\\$} \{0, 1\}^{nl}</math> 4 : <math>st \leftarrow (C_1, X_e, x_e)</math> 5 : <math>m_1 \leftarrow (C_1    X_e    EAD_1)</math> 6 : <b>return</b> <math>m_1</math> InitRun2(ID<sub>1</sub>, <math>x_s, (Y_e, c_2, C_R)</math>) ----- 1 : <math>PRK_{2e} \leftarrow RO_T(TH_2, Y_e^{x_e})</math> 2 : <math>TH_2 \leftarrow \mathcal{H}(Y_e, C_R, \mathcal{H}(m_1))</math> 3 : <math>sk_2 \leftarrow RO_P(PRK_{2e}, 0, TH_2, \ell_2)</math> 4 : <math>m_2 \leftarrow \mathcal{D}(sk_2, c_2)</math> 5 : <math>(ID_R    t_2    EAD_2) \leftarrow m_2</math> 6 : <math>CTX_2 \leftarrow (ID_R    TH_2    Y_s    EAD_2)</math> 7 : <math>salt_{3e2m} \leftarrow RO_P(PRK_{2e}, 1, TH_2, \ell_{hash})</math> 8 : <math>PRK_{3e2m} \leftarrow RO_T(salt_{3e2m}, Y_s^{x_e})</math> 9 : <math>t_2 \leftarrow RO_P(PRK_{3e2m}, 2, CTX_2, \ell_{mac})</math> 10 : <b>if</b> <math>t'_2 \neq t_2</math> : 11 :   <math>status \leftarrow rejected</math> 12 :   <b>return</b> <math>\perp</math> 13 : <math>TH_3 \leftarrow \mathcal{H}(TH_2, m_2)</math> 14 : <math>sk_3 \leftarrow RO_P(PRK_{3e2m}, 3, TH_3, \ell_{id})</math> 15 : <math>IV_3 \leftarrow RO_P(PRK_{3e2m}, 4, TH_3, \ell_{iv})</math> 16 : <math>salt_{4e3m} \leftarrow RO_P(PRK_{3e2m}, 5, TH_3, \ell_{hash})</math> 17 : <math>PRK_{4e3m} \leftarrow RO_T(salt_{4e3m}, Y_e^{x_s})</math> 18 : <math>sk'_3 \leftarrow RO_P(PRK_{4e3m}, 3, TH_3, \ell_{key})</math> 19 : <math>status \leftarrow accepted</math> 20 : <math>CTX_3 \leftarrow (ID_1    TH_3    X_s    EAD_3)</math> 21 : <math>t_3 \leftarrow RO_P(PRK_{4e3m}, 6, CTX_3, \ell_{mac})</math> 22 : <math>m_3 \leftarrow ID_1</math> 23 : <math>m'_3 \leftarrow (t_3    EAD_3)</math> 24 : <math>c_3 \leftarrow \mathcal{E}(sk_3, m_3)</math> 25 : <math>c'_3 \leftarrow \mathcal{E}'(sk'_3, IV_3; m'_3; c_3)</math> 26 : <math>TH_4 \leftarrow \mathcal{H}(TH_3, m_3    m'_3)</math> 27 : <math>PRK_{out} \leftarrow RO_P(PRK_{4e3m}, 7, TH_4, \ell_{hash})</math> 28 : <math>status \leftarrow terminated</math> 29 : <math>SK \leftarrow PRK_{out}</math> 30 : <b>return</b> <math>c_3</math> </pre>	<pre> Run(ID, sk, peerpk, m) ----- 1 : <b>if</b> <math>status \neq running</math> : 2 :   <b>return</b> <math>\perp</math> 3 : <b>if</b> <math>role = initiator</math> : 4 :   <math>m' \leftarrow InitRun2(ID, sk, m)</math> 5 : <b>elseif</b> <math>sid = \perp</math> : 6 :   <math>m' \leftarrow RespRun1(ID, sk, m)</math> 7 : <b>else</b> : 8 :   <math>m' \leftarrow RespRun2(ID, peerpk, m)</math> 9 : <b>return</b> <math>m</math> RespRun1(ID<sub>R</sub>, <math>y_s, m_1 = (X_e, C_1, EAD_1)</math>) ----- 1 : <math>y_e \xleftarrow{\\$} \mathbb{Z}_p</math> 2 : <math>Y_e \leftarrow g^{y_e}</math> 3 : <math>C_R \xleftarrow{\\$} \{0, 1\}^{nl}</math> 4 : <math>sid \leftarrow (C_1, C_R, X_e, Y_e)</math> 5 : <math>PRK_{2e} \leftarrow RO_T(TH_2, X_e^{y_e})</math> 6 : <math>sk_2 \leftarrow RO_P(PRK_{2e}, 0, TH_2, \ell_2)</math> 7 : <math>salt_{3e2m} \leftarrow RO_P(PRK_{2e}, 1, TH_2, \ell_{hash})</math> 8 : <math>PRK_{3e2m} \leftarrow RO_T(salt_{3e2m}, X_e^{y_s})</math> 9 : <math>TH_2 \leftarrow \mathcal{H}(Y_e, C_R, \mathcal{H}(m_1))</math> 10 : <math>CTX_2 \leftarrow (ID_R    TH_2    Y_s    EAD_2)</math> 11 : <math>t_2 \leftarrow RO_P(PRK_{3e2m}, 2, CTX_2, \ell_{mac})</math> 12 : <math>m_2 \leftarrow (ID_R    t_2    EAD_2)</math> 13 : <math>c_2 \leftarrow \mathcal{E}(sk_2, m_2)</math> 14 : <b>return</b> <math>(Y_e, c_2, C_R)</math> RespRun2(ID<sub>R</sub>, peerpk, <math>c_3</math>) ----- 1 : <math>TH_3 \leftarrow \mathcal{H}(TH_2, m_2)</math> 2 : <math>sk_3 \leftarrow RO_P(PRK_{3e2m}, 3, TH_3, \ell_{id})</math> 3 : <math>ID_1 \leftarrow \mathcal{D}(sk_3, c_3)</math> 4 : <math>X_s \leftarrow peerpk[ID_1]</math> 5 : <math>salt_{4e3m} \leftarrow RO_P(PRK_{3e2m}, 5, TH_3, \ell_{hash})</math> 6 : <math>PRK_{4e3m} \leftarrow RO_T(salt_{4e3m}, X_s^{y_e})</math> 7 : <math>IV_3 \leftarrow RO_P(PRK_{3e2m}, 4, TH_3, \ell_{iv})</math> 8 : <math>sk'_3 \leftarrow RO_P(PRK_{4e3m}, 3, TH_3, \ell_{key})</math> 9 : <math>m'_3 \leftarrow \mathcal{D}'(sk_3, IV_3; c_3; c'_3)</math> 10 : <b>if</b> <math>m'_3 = \perp</math> : 11 :   <math>status \leftarrow rejected</math> 12 :   <b>return</b> <math>\perp</math> 13 : <math>(t_3    EAD_3) \leftarrow m'_3</math> 14 : <math>status \leftarrow accepted</math> 15 : <math>CTX_3 \leftarrow (ID_1    TH_3    X_s    EAD_3)</math> 16 : <math>t'_3 \leftarrow RO_P(PRK_{4e3m}, 6, CTX_3, \ell_{mac})</math> 17 : <b>if</b> <math>t'_3 \neq t_3</math> : 18 :   <math>status \leftarrow rejected</math> 19 :   <b>return</b> <math>\perp</math> 20 : <math>TH_4 \leftarrow \mathcal{H}(TH_3, m_3)</math> 21 : <math>PRK_{out} \leftarrow RO_P(PRK_{4e3m}, 7, TH_4, \ell_{hash})</math> 22 : <math>status \leftarrow terminated</math> 23 : <math>SK \leftarrow PRK_{out}</math> 24 : <b>return</b> <math>\perp</math> </pre>
---	--

Fig. 6. Formalized description of the three flows EDHOC protocol