A comprehensive, formal and automated analysis of the EDHOC protocol

Charlie Jacomme, Elise Klein, Steve Kremer, Maïwenn Racouchot

A comprehensive, formal and automated analysis of the EDHOC protocol

Charlie Jacomme
Inria Paris[*]

Elise Klein
Inria Nancy
Université de Lorraine

Steve Kremer
Inria Nancy
Université de Lorraine

Maïwenn Racouchot
Inria Nancy
Université de Lorraine

Abstract

EDHOC is a key exchange proposed by IETF's Lightweight Authenticated Key Exchange (LAKE) Working Group (WG). Its design focuses on small message sizes to be suitable for constrained IoT communication technologies. In this paper we provide an in-depth formal analysis of EDHOC–draft version 12, taking into account the different proposed authentication methods and various options. For our analysis we use the SAPIC$^+$ protocol platform that allows to compile a single specification to 3 state-of-the-art protocol verification tools (PROVERIF, TAMARIN and DEEPSEC) and take advantage of the strengths of each of the tools. In our analysis we consider a large variety of compromise scenarios, and also exploit recent results that allow to model existing weaknesses in cryptographic primitives, relaxing the perfect cryptography assumption, common in symbolic analysis. While our analysis confirmed security for the most basic threat models, a number of weaknesses were uncovered in the current design when more advanced threat models were taken into account. These weaknesses have been acknowledged by the LAKE WG and the mitigations we propose (and prove secure) have been included in version 14 of the draft.

1 Introduction

EDHOC (Ephemeral Diffie Hellman Over COSE) is a key exchange protocol designed by IETF's Lightweight Authenticated Key Exchange (LAKE) Working Group (WG) to be used on constrained devices in the IoT. This kind of environment leads to strict requirements regarding implementation or use: for instance, bandwidth is limited along with the memory and therefore the design must prioritize small message sizes. Given today's immense number of IoT devices, security concerns are of crucial importance.

The LAKE WG was formed in 2019 and the first draft, version 00, of EDHOC was published in July 2020. Since then several drafts have been released and version 12 of the draft [21] was issued in October 2021. EDHOC is a public-key based authenticated Diffie Hellman (DH) key exchange protocol. It allows for different authentication methods, either based on signatures or static long-term DH keys. It also supports a specific version aiming at Post-Quantum security, replacing the DH key derivation by a Key Encapsulation Mechanism (KEM).

Automated, symbolic protocol analysis is a successful approach for finding attacks or proving their absence. The approach can be traced back to the seminal work of Dolev and Yao [12] at the beginning of the 80's. In the so-called Dolev-Yao model the attacker has complete control over the network and can eavesdrop, intercept and inject any messages. Cryptography is however treated in a rather abstract way, sometimes referred to as the *perfect cryptography assumption*, but this abstraction significantly eases automation of the verification. State-of-the-art verification tools, such as PROVERIF [6] and TAMARIN [20], are indeed able today to scale up to industrial-size, deployed protocols. They have in particular been used successfully to find weaknesses in early versions of the 5G standard [2] and actively assisted the standardization process of TLS 1.3 [4, 10]. Following these successes of formal analysis, Vučinić et al. *invite the formal analysis community to study the protocol and contribute the results in both the symbolic and the computational model* in a short paper which summarizes the design of EDHOC [23].

Contributions. In this paper, we present a comprehensive formal analysis of the EDHOC protocol: we provide a detailed model of version 12 and perform an analysis that combines several recent developments in formal methods.

- *Detailed models of the protocol, properties and primitives* (Section 3). We provide a detailed formal specification of version 12 of EDHOC. Our models include the 4 possible authentication methods, the KEM based version, and several optional checks. We also formally state the security properties claimed by the designers, see

[*] This work was partly done while Charlie Jacomme was at the CISPA Helmholtz Center for Information Security.

Section 3.2. Finally, we enrich our model with several state-of-the-art advanced models of cryptographic primitives, notably more precise models for signatures [14], DH groups [11], and hashes [7] [1].

The latter aim to provide more realistic symbolic models of cryptography, as we explain in Section 3.3. To the best of our knowledge this is the first case-study to combine all of these advanced models.

- *Combination of tools*. We used the recent Sapic$^+$ protocol verification platform [8]. Sapic$^+$ uses the applied pi-calculus as a specification language, which can then be exported into several security tools (Section 3.1). In our analysis (Section 4), we exported and proved the model first with ProVerif [6] which allows generally for faster results, then we used Tamarin [20] which allows for more precise modeling of DH exponentiation and hash functions. We moreover exploited Tamarin's interactive mode to find a hash transcript collision attack. In addition, we exported the model to DeepSec [9] for analyzing an anonymity property.

- *Methodology for a modular and comprehensive analysis* (Section 4.1). We followed a methodology that allowed us to analyze the protocol for a large combination of threat models, and protocol variants: we consider different threat models (key compromise and leakage), and optional protocol checks for each security property and systematically explore all of them, identify minimal threat scenarios for properties to hold, as well as maximal threat scenarios that violate a property. Moreover, all models are generated from a single file to increase maintainability, in particular to allow easy adaptation to future versions.

- *Attacks and mitigations* (Sections 4.2 to 4.7). Applying our methodology on version 12 of EDHOC, we highlight several weaknesses in the design and propose mitigations. All our findings were communicated to the LAKE WG and acknowledged. Possible mitigations were discussed and included in the latest version 14[2] of EDHOC, published in May 2022.

- *First results on version 14* (Section 4.8). We applied our methodology on version 14 and easily obtained a complete analysis of this version by leveraging the analysis pipeline set up for version 12. Our analysis confirms the security of the proposed fixes and ensures that they do not trigger other flaws.

Related Work. Formal verification techniques have been applied to many Internet standards and deployed protocols, including the authentication protocols of the 5G-AKA standard [2], TLS 1.3 [4, 10], and the SIGNAL secure messaging protocols [17] to name only a few.

Bruni et al. [19] performed a first formal analysis of EDHOC on draft 00 using the Tamarin prover. Their results and proposals were taken into account by the WG in draft 05. This analysis is however outdated as the protocol significantly changed since then. Further, it did not cover the 4 authentication methods at the same time, nor the KEM-based variant for Post-Quantum security. Also, the analysis in [19] did not verify anonymity properties. In [8], Sapic$^+$ was used to analyse EDHOC draft 07. This model was rather simple and did not cover the 4 authentication methods nor the KEM-based variant and the analysis did not result into any feedback to the WG. Moreover, the symbolic models of [8, 19] both assumed perfect cryptography as usual in symbolic models, and covered less key compromise scenarios. As we demonstrate, this assumption can miss significant attacks. We therefore enrich our model with state-of-the-art advanced primitive models [7, 11, 14] as explained before, as well as a large number of possible compromise scenarios. These extensions were indeed crucial to uncover many of the weaknesses.

A systematic exploration of different compromise and threat scenarios using symbolic tools has been performed in several other works. Basin and Cremers [3] formalize and explore different notions of key and state compromise in authenticated key exchange protocols. A similar methodology has been applied to the protocols form the Noise framework [13] and second-factor authentication protocols [16] to automatically compute the strongest threat model under which a protocol is secure. We are the first to combine the Sapic$^+$ methodology of leveraging multiple tools with such a systematic threat model analysis, as well as the combinations of advanced primitive models as was done in [7] for the particular case of hashes.

Outline. We present version 12 of EDHOC in Section 2 along with its claimed properties. We then explain how we model the protocol, properties and weaknesses in cryptographic primitives in Section 3. Finally, we describe our methodology and results in Section 4, before concluding.

Models and reproducibility. All our formal models are available online at [15], along with the source code of the Tamarin version that includes the Sapic$^+$ platform needed to run them. Standard versions of the DeepSec and ProVerif tools are sufficient and not provided at [15]. Alternatively, we provide a docker with the complete tool chain preinstalled that can be obtained and browsed using the following commands:

```
$ docker pull protocolanalysis/lake-edhoc:draft-14
$ docker run -it protocolanalysis/lake-edhoc:draft-14 bash
```

[1] In agreement with the chairs, reference [7] has been anonymized as a major revision is under submission at USENIX Security'23. The chairs will share [7] with the reviewers upon request.

[2] Draft 13 is a duplicate of draft 12 upon its expiration.

Initiator Responder	SIG	STAT
SIG	Method 0	Method 2
STAT	Method 1	Method 3

Table 1: Authentication methods usable with EDHOC

2 Presentation of the EDHOC protocol

EDHOC is a MAC-then-Sign Diffie Hellman key exchange protocol designed to be suitable for IoT usage. Authentication is provided using long-term public keys: the initiator I and the responder R can each either use a static long-term DH key (STAT) or a public signature key (SIG), resulting in 4 possible *methods* (Table 1). The protocol can be completed in three messages but allows an optional fourth message for additional security, i.e., key confirmation for the initiator.

2.1 Protocol outline

The EDHOC protocol consists of 4 messages, the last one being optional (and expected to be replaced by one at the application layer):

1. I sends session setup information along with its identifiers and public ephemeral key share;
2. R responds with its own identifiers, public ephemeral key share and authenticates its credentials;
3. I authenticates its credentials;
4. Optionally, R explicitly confirms computation of the session key.

We now explain the protocol in more details. As an illustration, we display in Figure 1 the method 0 version, i.e., when both the initiator and responder use signature keys, without the optional 4th message. We suppose that the initiator I and responder R have long-term signature keys ($\mathsf{sk}_I, \mathsf{pk}_I$ and $\mathsf{sk}_R, \mathsf{pk}_R$) or long-term, static DH keys (I, g^I and R, g^R).

Pseudo-random keys. Before detailing each message, we introduce the intermediary pseudo-random keys (PRK) that are computed during each session from the ephemeral shared key (and possibly static DH shares) and used to derive encryption and MAC keys:

- PRK_{2e} — encryption in 2nd message;
- PRK_{3e2m} — encryption in 3rd message and MAC in 2nd message;
- PRK_{4x3m} — encryption in 4th message and of application data and MAC in 3rd message. This is also the final key stored by both parties.

How each PRK is computed is method dependent and summarized in Table 2.

Meth.	PRK_{2e}	PRK_{3e2m}	PRK_{4x3m}
0	$\texttt{kdf}(G^{XY})$	PRK_{2e}	PRK_{3e2m}
1		$\texttt{kdf}(\mathrm{PRK}_{2e}, G^{XR})$	
2		PRK_{2e}	$\texttt{kdf}(\mathrm{PRK}_{3e2m}, G^{IY})$
3		$\texttt{kdf}(\mathrm{PRK}_{2e}, G^{XR})$	

Table 2: Derivation of intermediary keys.

First message. The initiator sends a message to the responder with the setup information (the method to be used, and the proposed cipher suite, i.e., a set of algorithm to be used for hashes, encryption, *etc.*), the public ephemeral key share G^X, a connection identifier C_I (with no cryptographic purpose) and some external authorization data EAD_1 (whose precise format is application dependent and left unspecified in [21]).

Second message. In case R agrees on the method and proposed cipher suite, R generates an ephemeral key share Y and computes the shared ephemeral key G^{XY}, and a transcript hash TH_2 by hashing the first message, its ephemeral key G^Y and connection identifier C_R. Basically, R responds by sending its ephemeral public key share G^Y. Authentication is ensured by a mac MAC_2 (with PRK_{3e2m}) and (in methods 0 or 2) an additional signature (with sk_R) on relevant data (pk_R, MAC_2, TH_2, and external authorization data EAD_2). Note that when only a mac is used, PRK_{3e2m} is derived from G^{XR}, i.e., involving the static, long-term DH key share R. The signature or mac is moreover encrypted using exclusive or with a key derived from PRK_{2e}.

Third message. Once $\mathsf{message}_2$ is received, I computes the session key G^{XY} and the decryption key derived from PRK_{2e}. The signature or mac in $\mathsf{message}_2$ allows verifying R's credentials. To construct the third message I computes a mac MAC_3 on the new transcript hash TH_3, on its credentials (pk_I or G^I) and external data EAD_3 using *PRK_4x3m*. MAC_3, along with its content is then protected (depending on the method) by an additional signature and encrypted using an authenticated encryption scheme ($\texttt{aead}$) with a key derived from PRK_{3e2m}.

When R receives $\mathsf{message}_3$, R will decrypt it and verify the signature or mac to authenticate the initiator. At this point, the protocol can stop with the session key PRK_{4x3m} shared between both participants.

Fourth message. The responder may send back a final message containing external authorized data encrypted with the session key to act as an acknowledgment.

A post-quantum secure variant. It is noted in [21] that the proposed protocol is not post-quantum secure, in particular due to the use of a DH key exchange, relying on the hardness of the discrete logarithm. It is suggested that the

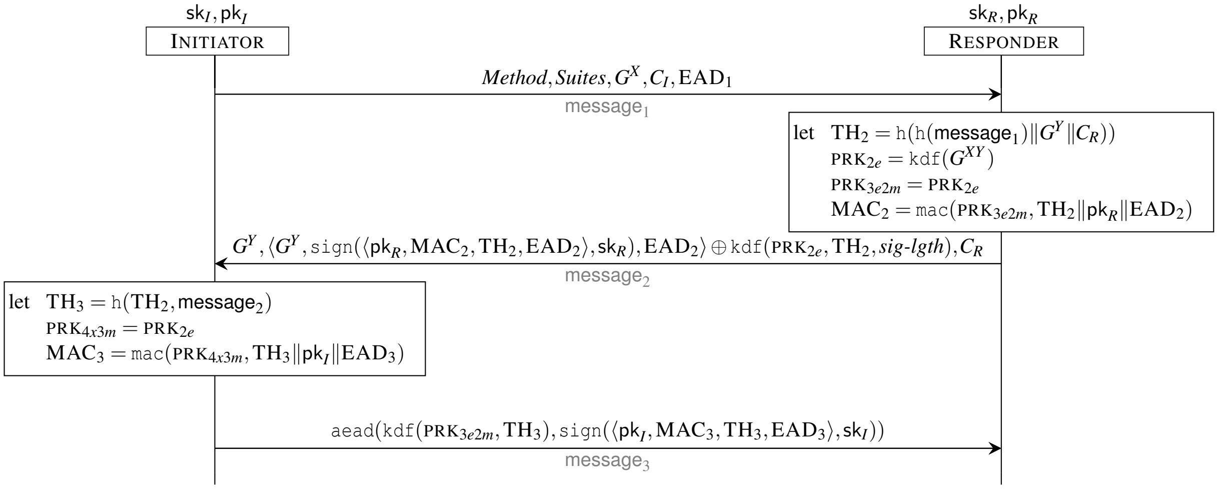
$\mathsf{sk}_I, \mathsf{pk}_I$
INITIATOR
$\mathsf{sk}_R, \mathsf{pk}_R$
RESPONDER
$Method, Suites, G^X, C_I, \mathrm{EAD}_1$
$\mathsf{message}_1$
let $\mathrm{TH}_2 = \mathtt{h}(\mathtt{h}(\mathsf{message}_1) \| G^Y \| C_R)$
$\mathrm{PRK}_{2e} = \mathtt{kdf}(G^{XY})$
$\mathrm{PRK}_{3e2m} = \mathrm{PRK}_{2e}$
$\mathrm{MAC}_2 = \mathtt{mac}(\mathrm{PRK}_{3e2m}, \mathrm{TH}_2 \| \mathsf{pk}_R \| \mathrm{EAD}_2)$
$G^Y, \langle G^Y, \mathtt{sign}(\langle \mathsf{pk}_R, \mathrm{MAC}_2, \mathrm{TH}_2, \mathrm{EAD}_2 \rangle, \mathsf{sk}_R), \mathrm{EAD}_2 \rangle \oplus \mathtt{kdf}(\mathrm{PRK}_{2e}, \mathrm{TH}_2, sig\text{-}lgth), C_R$
$\mathsf{message}_2$
let $\mathrm{TH}_3 = \mathtt{h}(\mathrm{TH}_2, \mathsf{message}_2)$
$\mathrm{PRK}_{4x3m} = \mathrm{PRK}_{2e}$
$\mathrm{MAC}_3 = \mathtt{mac}(\mathrm{PRK}_{4x3m}, \mathrm{TH}_3 \| \mathsf{pk}_I \| \mathrm{EAD}_3)$
$\mathtt{aead}(\mathtt{kdf}(\mathrm{PRK}_{3e2m}, \mathrm{TH}_3), \mathtt{sign}(\langle \mathsf{pk}_I, \mathrm{MAC}_3, \mathrm{TH}_3, \mathrm{EAD}_3 \rangle, \mathsf{sk}_I))$
$\mathsf{message}_3$

Figure 1: EDHOC protocol, method 0: both initiator and responder use signature keys

protocol can be made post-quantum secure by relying on a *key encapsulation mechanism* (KEM): the resulting protocol is obtained using method 0 (instantiated with post-quantum secure signatures), and replacing the DH by a KEM-based key exchange.

2.2 Claimed properties

In [21], a number of properties are claimed that we present in this section and formalize and analyze later.

2.2.1 Confidentiality

We have identified four confidentiality properties. From the responder point of view, a completed EDHOC exchange should provide confidentiality of the key, and the responder is also ensured that the initiator computed the same key.

Explicit Key Confirmation **[21, p. 44]**

After verifying $\mathsf{message}_3$, the Responder is assured that the Initiator has calculated the key PRK_{4x3m} (explicit key confirmation) and that no other party than the Responder can compute the key.

However, from the initiator point of view only the confidentiality part is ensured as the fourth message is optional.

Implicit Key Authentication **[21, p. 44]**

After sending $\mathsf{message}_3$, the Initiator is assured that no other party than the Responder can compute the key PRK_{4x3m}.

Confidentiality of the final session key must hold under strong forms of compromise: either compromise of other session keys or future compromise of the long-term keys.

Session key independence **[21, p. 43]**

Compromise of one session key does not compromise other session keys.

Forward secrecy **[21, p. 43]**

Compromise of the long-term keys (private signature or static DH keys) does not compromise the security of completed EDHOC *exchanges.*

2.2.2 Authentication

In addition to the Explicit Key Confirmation, which has already a flavor of authentication, EDHOC should also guarantee authentication of some of the exchanged data.

Authenticated transcript hash **[21, p. 8]**

Transcript hashes (hashes of message data) TH_2, TH_3, TH_4 (are) used for key derivation and as additional authenticated data.

Authenticated data **[21, p. 42]**

EDHOC *adds an explicit method type and expands the message authentication coverage to additional elements such as algorithms, external authorization data, and previous messages.*

Similar to confidentiality, some authentication should also be preserved even in case of compromises.

Key compromise impersonation **[21, p. 43]**

Compromising the private authentication keys of one party lets an active attacker impersonate that compro-

> *mised party in* EDHOC *exchanges with other parties but [...] does not let the attacker impersonate other parties in* EDHOC *exchanges with the compromised party.*

2.2.3 Identity protection

EDHOC has also been designed to offer anonymity guarantees, but mostly to the initiator. Indeed, a responder obviously leaks its identity to any initiator.

> **Identity protection** **[21, p. 42]**
>
> EDHOC *protects the credential identifier of the Initiator against active attacks and the credential identifier of the Responder against passive attacks.*

2.2.4 Non-repudiation

Depending on the method, a participant should not be able to deny having participated in a session. The draft specifies that, the other user could present to a judge its own ephemeral private key as an evidence. Note that this implies storing the ephemeral keys, therefore violating some requirements of the protocol.

> **Non-repudiation** **[21, p. 44]**
>
> *In* EDHOC *authenticated with signature keys, the Initiator could theoretically prove that the Responder performed a run of the protocol by presenting the private ephemeral key, and vice versa.*

3 Models for automated verification

3.1 Symbolic tools

For our analysis we use the SAPIC$^+$ protocol verification platform [8]. SAPIC$^+$ allows to provide a single protocol specification and use three state-of-the-art protocol verification tools as backends: PROVERIF, TAMARIN and DEEPSEC. Therefore, the strengths of each of these tools can be exploited. Moreover, SAPIC$^+$'s correctness proof allows reusing results proved in one tool in the other tools. In this section we give an overview of how protocols and properties are specified in SAPIC$^+$, using examples and informal explanations; the interested reader can find a formal treatment of the complete syntax and semantics in [8].

Messages as terms. As usual in symbolic models, messages are represented as *terms*: in contrast to concrete bitstrings, terms focus on the message structure, abstracting away implementation details. Atomic terms are used to represent fresh values to model, e.g., a randomly sampled secret key or nonce, as well as variables which represent unknowns, typically used in the inputs of a protocol specification and instantiated during executions. Cryptographic operations are represented using *function symbols*. For example, digital signatures can be modeled by the four following function symbols:

$$\mathtt{sign}(\cdot,\cdot) \qquad \mathtt{verify}(\cdot,\cdot,\cdot) \qquad \mathtt{vk}(\cdot) \qquad \mathtt{true}$$

where `true` is a constant, i.e., does not take any argument.

Terms are either atomic or constructed by applying function symbols to other terms. For instance, the term $\mathtt{sign}(m, sk)$ may represent inside the run of a protocol the signature of a term m with the private signing key sk.

To actually express that function symbols represent a given cryptographic primitive we define their properties using an equational theory. For example, the equation

$$\mathtt{verify}(\mathtt{sign}(x,y),x,\mathtt{vk}(y)) = \mathtt{true}$$

expresses that the verification of a signature of $\mathsf{message}_x$ with private signing key y returns `true` when the verification algorithm is given the matching $\mathsf{message}_x$ and corresponding verification key $\mathtt{vk}(y)$. Importantly, equations specify the *only* equalities that hold, i.e., the verification algorithm never returns `true` otherwise. This hypothesis is sometimes referred to as the "perfect cryptography assumption", but as we will explain in Section 3.3, we will relax this assumption, relying on recent results.

Protocols as processes. Protocols are modeled as processes in a dialect of the applied pi-calculus [1]. The applied pi-calculus is a specification language for modeling cryptographic protocols as communicating, parallel programs. Its semantics takes into account that communications happen over an adversary-controlled network: the adversary can intercept any message and insert forged messages. To forge a message an adversary may build terms from public atoms and previously intercepted messages, possibly applying function symbols on these.

Giving a full, formal description of the language is beyond the scope of this paper. We rather explain the main features on a few examples. The following snippet, for instance, represents a simplified extract of the process modeling the initiator role.

```
let Init(skI, I) =
    in(<method,suitesI,ID_CRED_R,C_I,EAD_1>);
    new X;
    event Share(X);
    let G_X = g^X in
    let G_I = g^I in
    let m1 = <method,suitesI,G_X,C_I,EAD_1> in
    out(m1);
    [...]
```

The initiator takes the private signing key `skI` and the long-term DH share `I` as parameters. Next the initiator inputs from the network a tuple of values with the `in` instruction. These values define the protocol method and cipher suite to be used, as well as several other parameters. ID_CRED_R identifies the credentials of the responder the initiator is going to engage with, and C_I and EAD_1 are values that are unspecified by the protocol. This initial input is actually provided by the attacker and a classical trick to model worst case scenarios, by letting the adversary choose the responder, methods, and values that would suit it most. Next, with the **new** instruction, we generate a fresh, ephemeral DH share X. Intuitively, **new** X models the sampling of a fresh random value, unknown to the attacker, that is then stored in X. We use an **event** instruction to log that X is an ephemeral key share. Such an **event** does not influence the execution, but is simply an annotation that we use to specify security properties as we will discuss below. The initiator then computes the public key shares G_X and G_I for the ephemeral, respectively long-term keys. Here, g^a models exponentiation of the group generator g and comes with a built-in equational theory, expressing properties such as (g^a)^b = (g^b)^a. Finally we prepare the tuple m1 to be output as the first protocol message. Additionally the language provides conditionals to verify that received messages are as expected as in the following snippet where the initiator checks the signature when method 0 is chosen.

```
if verify(SIGNATURE_or_MAC_2,<'Signature1',
  ID_CRED_R,TH_2,pkR,EAD_2,MAC_2>,pkR) = true
  then
    [...]
```

Finally, we need to explain how the different roles are run in parallel. Consider the following main process.

```
!( new sk; new ltdh;
    ( !Init(sk, ltdh)) | ( !Resp(sk,ltdh) ) )
```

Replication, denoted **!**, allows to spawn an unbounded number of concurrent copies of a process. Here, we can spawn an arbitrary number of participants. For each participant we generate long-term keys sk and ltdh and the participant can engage in an unbounded number of sessions, either as an initiator or a responder where | denotes the parallel composition.

Properties as first-order formulas. Security properties are expressed as first-order logic formulas over events that must hold on all possible execution of the protocol. Again, rather than providing the formal syntax and semantics we provide an example to introduce the most salient features of the logic.

```
∀ pkI pkR k4 Y GX i j k.
AcceptR(pkI,pkR,k4,Y,GX)@i & Honest(pkI)@k
 ⇒ not K(k4)@j
```

This property states a simple version of confidentiality of a key. The event **AcceptR** is a process annotation indicating that the responder has accepted a session between two participants (identified here by their public keys pkI and pkR) and that he believes they established a key k4. The event **Honest** (pkI) indicates that pkI was indeed honestly generated, i.e., not generated by the attacker. Indeed, if the initiator were the attacker, confidentiality of the established key would be trivially broken. (Note that we do not need to require that the responder is honest, as the **AcceptR** event cannot occur in an adversary process.) **K** is a built-in event that models the attacker knowledge, i.e., **K**(t) is true whenever the attacker can deduce the term t from previously received or intercepted messages. In the above formula i, j, k are *timepoints* and the syntax @i means that the event happened at timepoint i, but those timepoints were not exploited in this property. We illustrate their use on the following simple authentication property.

```
∀ pkI pkR k4 Y GX i k.
AcceptR(pkI,pkR,k4,Y,GX)@i & Honest(pkI)@k
⇒ (∃ t X GY k3. t<i &
                AcceptI(pkI,pkR,k3,k4,X,GY)@t)
```

This property expresses that each time the responder accepts a session (with an honest initiator), then the initiator must have accepted a matching session before. The fact that the sessions match is expressed by the fact that relevant event parameters, e.g., pkR, pkI, k4, coincide. The fact that the initiator accepted *before* is explicit by requiring t<i. More advanced properties, taking into account different key compromise scenarios, will be discussed below.

Finally, properties may sometimes be stated as *equivalences*. Intuitively, when two processes P and Q are equivalent, written $P \approx Q$ it means that for any *arbitrary* process A (the attacker process), $A \mid P$ and $A \mid Q$ exhibit the same behaviour. In other words an arbitrary attacker running in parallel cannot distinguish whether they interact with P or with Q. Such equivalences are useful to model anonymity properties.

3.2 Protocol and properties modeling

3.2.1 Protocol model

Overview of the protocol model. Our model takes into account an unbounded number of participants. As shown in the code snippet of the previous section, each participant can engage into multiple protocol sessions, acting either as the initiator, or the responder. Moreover, we model the four possible methods, depending on whether long-term signature keys or long-term DH keys are used for authentication. As explained above, the adversary chooses the intended communication partner for the initiator, and the method to be used. The attacker can also generate its own key material and choose to participate in a given session. Therefore, our model includes

- sessions between honest participants, in particular a same agent taking both the role of the initiator and the responder; hence, reflection or selfie style attacks are in the scope of the model;
- sessions where an honest initiator engages with the attacker;

- sessions where the attacker takes the role of the initiator and engages with a honest responder.

We additionally modeled the KEM-based variant: in method 0, the DH based key exchange can be replaced by a KEM-based one. Indeed, post-quantum security could be obtained by choosing both the signature and KEM schemes to be post-quantum secure.

Key compromise. We model a variety of (dynamic) key compromise scenarios. At any moment, the attacker can decide to compromise a key. In our model this is achieved by making compromise events available as parallel processes. We consider the following events, followed by the secret key outputs.

- `Compromise(k)`: compromise of either the long-term signature or long-term DH key;
- `LeakShare(x)`: compromise of the ephemeral DH key share; note, that for convenience we always raise two events, one with argument `g^s` and one with argument `s`, but we always leak the secret part `s` of the share;
- `LeakSessionKey(key)`: compromise of the session key PRK_{4x3m}.

As we will see below, our security properties will be conditional on whether some keys have been compromised.

Limitations. Currently, our model lacks some features. We do not model the *cipher suite negotiation*. The suite is simply a constant chosen by the attacker. However, we do verify authentication and agreement of the chosen suite. Our model also omits the *key update mechanism*, which allows achieving more efficient forward secrecy by updating the session key PRK_{4x3m} rather than re-running a new protocol session. Finally, our model does not include the optional fourth message, whose purpose is to provide explicit key confirmation to the initiator. These extensions would add complexity to our already rather extensive and large existing models.

3.2.2 Confidentiality properties

We analyse the 4 confidentiality properties discussed in Section 2.2.1. Interestingly, we are able to state a single strong confidentiality property for each of the participants that implies all 4 properties. We here state the property for the initiator. (The property for the responder is stated in a similar way.)

```
∀ pkR k3 k4 X GY  i j k.
(AcceptI(pkI,pkR,k3,k4,X,GY)@i & KU(k4)@j
 & Honest(pkR)@k)
  ⇒ (∃ t. ( Compromise(pkR)@t & t < i) ) |
     (∃ t. LeakSessionKey(k4)@t) |
     (∃ t. LeakShare(GY)@t) |
     (∃ t. LeakShare(X)@t)
```

Intuitively, the property states that *if the attacker learns the key established (according to I) with a honest responder then the responder's long-term key was compromised before or the session key was leaked or the responder share was leaked or the initiator share was leaked.*

This property indeed encodes all 4 confidentiality properties. *Forward secrecy* is encoded by requiring the initiator's long-term key to be compromised *before* the key is accepted by the initiator. In other words, if the compromise happens after the key is accepted, and the attacker learns the session key, the property is violated (or one of the other disjunction holds).

Session key independence is ensured as we only consider leakage of the current session key `k4`. Leakage of session keys from different sessions does not directly satisfy the property, and so the attacker is free to leak them. As the ephemeral key shares allow to recompute the session keys (at least in some methods) they are handled as the session key.

Finally, the above property encodes *Implicit Key Authentication* because the `AcceptI` is placed next to sending $\mathsf{message}_3$. Similarly, in the responder version, the `AcceptR` event is placed after having verified $\mathsf{message}_3$ to ensure *Secrecy after Explicit Key Confirmation*.

3.2.3 Authentication properties

Similarly, we model authentication properties. For instance, the following property states entity authentication for the responder, and explicit key confirmation.

```
∀ pkI pkR k4 Y GX i k.
AcceptR(pkI,pkR,k4,Y,GX)@i & Honest(pkI)@k
 ⇒(∃ t X GY k3. t<i &
              AcceptI(pkI,pkR,k3,k4,X,GY)@t) |
   (∃ t. Compromise(pkI)@t) |
   (∃ t. LeakShare(Y)@t)
```

We basically state that each time the responder `pkR` accepts a session with initiator `pkI`, resulting in session key `k4`, the initiator previously accepted a session with the same parameters, or the attacker compromised keys that break the property trivially. Note that this property also implies KCI, as an attacker can compromise `pkR` without directly satisfying the property.

One may note that this is a non-injective authentication property: as stated the property does not protect against a replay attack, i.e., a responder might accept two sessions while the initiator accepted only one. Therefore we additionally prove uniqueness of the responder's accept event, i.e., there are no two accept events for the same initiator, responder, and session key. This is a stronger property that entails injectivity of the authentication. A symmetric authentication property is shown for the initiator.

In addition, we also prove *transcript authentication* and authentication of other data such as algorithms, external authorization data, and previous messages. The modeling is the same as above except that the accept events are replaced by

`AcceptRData(...)` and `AcceptIData(...)` which contain arguments `THx,EAD_x,mx` that represent the transcript hashes, the external authorization data, and previous messages.

3.2.4 Identity protection

We can model identity protection as an *anonymity* property, expressed as an equivalence $P_1 \approx P_2$ where P_b is defined as

```
! ( new sk1; out(pk(sk1)); new sk2; out(pk(sk2));
    ! in(pkR); Init(sk1,pkR) |
    ! in(pkR); Init(sk2,pkR) |
    ! Resp(sk1) | ! Resp(sk2) |
    ! Init(skb,pk(sk1))
  )
```

This process allows to create any two pairs of users. If we consider that `Init` takes as parameters its secret signing key and the responder public key it will contact, each user can act as the initiator starting a session with an attacker chosen responder `pkR` or as a responder. We finally add a test session that depends on the *challenge bit b*: `sk`b, i.e., either `sk1` or `sk2` initiates a *test session* with `sk1` (here, `sk1` is chosen arbitrarily; we could have chosen `sk2`). The equivalence expresses that the adversary cannot tell these two processes apart.

We note a few important details: in the test session the attacker may not choose the responder's identity. Otherwise, the attacker could act as the responder and trivially break the equivalence. Moreover, in our verification we forbid compromise of `sk1`, as otherwise, again, the attacker could act as the responder and trivially know b. Interestingly, we may however allow compromise of `sk2`.

3.2.5 Non-repudiation

Non-repudiation is more tricky to specify. The goal is to provide evidence to an external party (that we call a *judge*) that a designated party indeed participated in a session. The property can be split in two parts:

- *Soundness:* if evidence is accepted by a judge then the designated party did participate in the session.

- *Completeness:* if a party did participate in a session, then the other participant can present evidence to the judge that will be accepted.

We focus on soundness: this property should guarantee that the attacker cannot provide false evidence that the responder participated in a session. To show that the responder is guaranteed soundness, we use an additional judge process. The judge inputs a non-repudiation proof `proofnr` and checks its validity. If the check succeed, the judge raises an event `WasActiveR(pkr,derivedKey,proofnr)`: the proof `proofnr` claims that the responder `pkR` derived key `derivedKey`. We then verify the property:

```
∀ pkr derivedKey proofnr i j.
WasActiveR(pkr,derivedKey,proofnr)@i &
Honest(pkr)@j
 ⇒ (∃ k. DerivedRShared(pkr,derivedKey)@k) |
   (∃ k. Compromise(pkr)@k)
```

where `DerivedRShared(pkr,derivedKey)` indicates that the responder `pkr` computed the shared, ephemeral DH key `derivedKey`. A similar modeling is used for the initiator.

We also model an *injective* variant of this property: the judge will not accept two distinct evidences proving participation in a same session.

3.3 Advanced primitive modelings

As explained above, symbolic models generally idealize cryptography. However, recent work [7, 11, 14] has relaxed this idealization by explicitly modeling existing weaknesses that exist in concrete primitives. In this work, we aim at making our analysis as precise as possible, building on such more precise models whenever they apply to the concrete algorithms supported by EDHOC. We thus include state-of-the-art models of cryptographic primitives, and notably capture the following aspects.

Precise signature models (Sig$^{\lightning}$, Sig$^{\lightning}$-proof). The ES256 signature scheme is malleable, i.e., given a signature on a message it is possible to forge a different signature for the same message. The edDSA signature scheme allows to (dishonestly) generate keys for which the verification of a signature will always succeed. A symbolic modeling of these and other weaknesses, that is amenable to automation in TAMARIN, is proposed in [14]. In [14], two models are actually proposed: a first model explicitly adds equations for specific weaknesses, while the second allows any behaviour that does not contradict unforgeability assumptions of the signature. The first model, that we denote by Sig$^{\lightning}$, is better suited for attack finding; the second model aims at security proofs, and we refer to it as Sig$^{\lightning}$-proof.

Precise DH models (DH$^{\lightning}$). Similarly, more precise models have been proposed for DH exponentiation. First, there exists an identity element e in the ECDH group such that $e^x = e$ for all x. Moreover, in all curves supported in [21], there exist small subgroups of elements $(h_1, \ldots, h_n)$ [11] such that:

- many collisions of the form $h_1^x = h_2^y$ exist;

- given g^x, for all z we will have with non-negligible probability that $(h_1 g^x)^z = g^{xz}$.

Models for automating the verification with these additional capabilities have been proposed in [11] for TAMARIN. We refer to these modelings as DH$^{\lightning}$.

Precise hash functions ($\mathsf{Hash}^{\lightning}$). Modeling hash functions in the symbolic model is reminiscent of the random oracle model, which is unrealistic. Some hash functions allow for length extension, i.e., $h(x||y) = h(h(x)||y)$. There may also exist chosen-prefix collisions, where for all p_1, p_2, the attacker may be able to compute c_1, c_2 such that $h(p_1||c_1) = h(p_2||c_2)$, see [22] for a survey. Such weaknesses (in combination with length extension attacks) have been found to lead to real attacks on protocols, where [5] have exploited such weaknesses in SHA-1 to trigger transcript collision attacks over TLS. [21] supports SHA-256 that allows length extensions, but for which there is no feasible chosen-prefix collision attack yet. Nevertheless, we consider it valuable to future-proof the protocol against such weaknesses if they were to appear, especially in the context of IoT where it may be even more difficult to deprecate an algorithm. We also note that practitioners often consider second-preimage resistance to be a sufficient property. Chosen-prefix collisions do *not* contradict second-preimage collisions and it might therefore be tempting, e.g., for efficiency reasons, to consider hash functions that are only second-preimage resistant, but not necessarily collision resistant. In [7], a more precise symbolic model that accounts for length extension and chosen prefix collisions has been proposed and automated in both TAMARIN and PROVERIF. We refer to this precise hash model as $\mathsf{Hash}^{\lightning}$.

Precise encryption models ($\oplus^{\lightning}$, $\mathsf{AEAD}^{\lightning}$). The XOR encryption used in $\mathsf{message}_2$, which encrypts a triplet, is obviously malleable. We therefore added tailored equations that allow to modify each of the three elements. We denote this model by $\oplus^{\lightning}$. The AEAD used in $\mathsf{message}_3$ may only provide integrity of the plaintext and not of the ciphertext. This can be the case for the Mac-Then-Encrypt construction (even though not currently supported in [21]), or when the encoding and decoding of the message containing the encryption is not fully deterministic (implementation dependent). We therefore model the possibility to re-randomize an AEAD ciphertext, and denote this more precise model of AEAD as $\mathsf{AEAD}^{\lightning}$.

It is interesting to note that although the models from [14] and [11] were designed for TAMARIN, it was straightforward to port them to PROVERIF and fully integrate them inside the SAPIC^{+} platform.

4 Results of the analysis

Our analysis confirms the strong design of the EDHOC protocol as most claimed properties are satisfied when we consider *basic* threat models. However, when exploring more advanced threat models on detailed protocol models we discovered multiple weaknesses.

We present in the following the details of the identified weaknesses as well as mitigations to strengthen the protocol. While most weaknesses were found using automated analysis, we also manually identified some additional weaknesses. Those were notably discovered when exploring what were the limitations of our models w.r.t. to the draft, questioning each limitation leading to see which points of the standard required additional inspection.

Disclosure process. We reported each of the identified weakness to IETF's LAKE working group, along with recommendations and concrete proposals to mitigate them. The weaknesses were reported as git issues, followed by discussions and potential pull-requests. Due to the lack of up-to-date implementations of the draft, we could not verify the presence of these weaknesses in a concrete implementation. However, each weakness was discussed with the WG: most were found relevant and the discussions lead to the integration of major changes to the key derivation, and several other fixes in the editor version of the draft and later to draft 14.

4.1 Methodology

We describe here how we analysed the models described in Section 3, in a comprehensive and modular way. We also took into account maintainability of our models, which notably allowed for a swift update from the models of draft 12 to draft 14.

Modular threat models. As explained in Section 3, even in basic models, we consider a Dolev-Yao attacker that completely controls the network, and can manipulate messages. Moreover, the attacker chooses the parameters of honest agents' sessions: notably, the attacker decides the identity of the peer, and the attacker can actively participate using its own set of identities. The attacker can also compromise long term secret keys.

This core threat model is extended by multiple attacker capabilities in a modular way: compromise of ephemeral shares, denoted by $\mathsf{DHShare}^{\lightning}$, compromise of session keys denoted by $\mathsf{SessKey}^{\lightning}$, and the advanced primitive models of Section 3 are atomic capabilities that can be combined. Our aim is to explore the different security properties by considering all possible combinations of those attacker capabilities.

In addition, we also implement modularly two additional checks that the protocol *may* perform:

- DH-Check - Agents refuse incoming neutral DH element;
- Cred-Check - Agents refuse sessions with their own identity as the peer.

To implement DH-Check, an agent checks whether the peer's ephemeral key share, G^X or G^Y, is the identity element. Note that when using $\mathsf{DH}^{\lightning}$, DH-Check is insufficient to forbid the low order points of the elliptic curves, but we did not model a more advanced check as the WG preferred to avoid implementing even the basic DH-Check. Cred-Check is an optional

Attacker Capabilities	
Sig$^{\lightning}$	precise signature (for attack finding)
Sig$^{\lightning}$-proof	precise signature (stronger guarantees)
DH$^{\lightning}$	precise DH with small subgroups
AEAD$^{\lightning}$	rerandomizable cyphertexts model
Hash$^{\lightning}$	Hash with length-extension and chosen prefix collisions
$\oplus^{\lightning}$	malleable xor encryption
SessKey$^{\lightning}$	compromise of session keys
DHShare$^{\lightning}$	compromise ephemeral shares
Protocol Optional Checks	
DH-Check	neutral DH element check
Cred-Check	own identity check

Table 3: Summary of the atomic scenarios

check for trust on first use (TOFU) : the initiator should verify that the responder's identity is not equal to its own.

We summarize atomic attacker capabilities in Table 3. Our goal is to verify all possible combinations of these atomic capabilities. However, we can be more efficient and avoid the verification of redundant checks, similar to [7, 16]. The core idea is that the scenarios form a lattice ordered by inclusion, and proving that a property holds in a particular threat model implies that it also holds in all weaker scenarios. Conversely, if a property is violated inside a given threat model, say Hash$^{\lightning}$, it will also be violated in all stronger threat models, i.e., all threat models that contain Hash$^{\lightning}$. More redundant verifications can be avoided as for instance the Sig$^{\lightning}$capability is weaker than the Sig$^{\lightning}$-proof.

We aim to identify for each property the maximal, i.e., strongest, threat models for which the property hold, as well as the minimal, i.e., weakest, ones for which there is an attack while pruning any redundant verifications.

Using Sapic^{+} for maintainability. A core concern in our models was to factorize the code base as much as possible, both to avoid mistakes and ease maintainability with respect to updates of the draft. Our process relies on Sapic^{+}, the corresponding backends Proverif, Tamarin and DeepSec, and the templating engine Jinja2:

1. Using Jinja2, we generate from a single file both the DH and KEM based protocols. Moreover, for the DH version, all four methods are derived from a shared template.

2. Using Sapic^{+} and its basic preprocessing capabilities to enable or disable an attacker capability, we generate the file for a given list of atomic threat models and a target verification tool.

3. We finally perform the verification on the generated file using the corresponding tool.

Given a protocol variant, threat model, and a security property, a single line CLI allows to streamline this process and launch the verification. From this, we designed a few small scripts that batch run the verifications on all relevant threat model combinations.

Comprehensive verification with Sapic^{+}. As a benefit from using Sapic^{+}, we can choose the most suited tool for each verification: Proverif and Tamarin are used for authentication, confidentiality and non-repudiation properties (so called reachability properties), and DeepSec and Proverif for anonymity (modeled as an equivalence property). Overall, Proverif tends to be faster, while Tamarin offers a more precise model of DH exponentiation, that notably includes inverse group elements, and thus provides stronger guarantees. Even though we did not encounter this case in our analysis, this implies that for a given threat model Proverif could prove a property while Tamarin finds an attack.

While developing the model, we rely on Proverif to perform efficient sanity checks and ensure that the protocol executes correctly. Interestingly, the correction result of Sapic^{+} [8] ensures that these sanity checks obtained by Proverif (through Sapic^{+}) carry over to the Tamarin and DeepSec models. Hence we avoid running them in these other tools, which can be cumbersome.

To perform a comprehensive verification and explore all possible threat models, we then rely on the efficiency of Proverif to first quickly comb through all the scenarios. Whenever Proverif finds an attack in a given scenario, there is no need to check it with the more precise DH theory of Tamarin. For maximal scenarios in which a given property holds, we use Tamarin with a longer time out to strengthen the trust in this proof.

In more details, our global approach is as follows.

1. Run Proverif on all threat model combinations with a timeout of 30 minutes over both the DH and the KEM variants.

2. Extract minimal threat models required for an attack, and maximal ones for a security proof. This allows us to extract a small set of interesting scenarios that can be verified by a script we prepared for reproducibility with Proverif;

3. We finally use Tamarin with a long timeout of 24 hours to increase the trust on all maximal, i.e., strongest, threat models identified by Proverif in the previous step.

As anonymity properties are harder to check we do not consider any of the advanced primitive models and first verify the KEM version for a bounded number of sessions with DeepSec. (DeepSec does not support DH exponentiation nor unbounded number of sessions.) If the property holds, we verify it for an unbounded number of sessions with PROVERIF for both the KEM and DH protocols (but restricted to method 0 for efficiency reasons).

This full process was originally developed for draft 12 and allowed us to report several weaknesses and propose improvements to the WG, most of which were integrated into draft 14. We were then able to verify draft 14 very efficiently, by *(i)* updating a single model, *(ii)* running the batch PROVERIF script of step 2, and *(iii)* the batch TAMARIN script of step 3.

Note that the same analysis and methodology could be done without SAPIC$^+$ by writing manually tool specific models. However, SAPIC$^+$ greatly sped up the analysis, e.g., it only required writing a single model (while the modeling languages of TAMARIN and PROVERIF/DEEPSEC fundamentally differ), sanity checks of the model are significantly faster in PROVERIF. In particular, using only PROVERIF, the DH model would have been less precise; using only TAMARIN would complicate the anonymity analysis (support for equivalence, in particular in the presence of restrictions, is less mature in TAMARIN) and exploration of the whole threat model lattice (as verification takes more time). Finally, it is likely that reaching our level of maintainability would be impossible with separate models.

We believe that the infrastructure we developed is of independent interest and could be reused out of the box for other protocol analyses. For instance, our models are split into multiple parts with separate header files for easy reuse of the advanced primitive models.

4.2 Summary of the automated analysis of draft 12

We summarize in Table 4 highlights of our analysis showing secure and insecure scenarios. Overall, most properties hold in the core threat model but we found multiple weaknesses in advanced threat models for which we propose fixes to strengthen the protocol. We provide in Table 5 a list of the main attacks we identified. We also specify the required threat model, the tool used to find the attack and the verification time. Moreover, we indicate whether the weakness was fixed in the draft 14 following our proposals. The extensive results of our analysis, specifying the exact lemmas and considered threat models, on both draft 12 and 14, can be found in Appendix B.

All our experiments were performed on a 64 core Intel(R) Xeon(R) CPU E5-4650L 0 @ 2.60GHz server with 756GB of RAM. Overall, the initial batch verification on draft 12 required 273 single threaded PROVERIF calls for an accumulated runtime of 20.6 hours (verified in less than an hour with a 64 core parallelization). We extract from this batch verification the 50 maximal and minimal threat scenarios, that correspond to 3h of PROVERIF accumulated runtime.

4.3 Weaknesses in the Key Derivation

In [21], the final key material consists in the key PRK_{4x3m}, built from the ephemeral shared DH secret, and the final transcript hash TH_4. An optional key exporter is suggested that binds PRK_{4x3m} and TH_4. However, several attack traces found by our tool illustrate weaknesses in this key derivation.

Leaking ephemeral secrets breaks authentication. Although one might expect authentication to rely only on the secrecy of the long-term keys, a violation of entity authentication was reported by our tool when we compromise both the ephemeral DH shares and the session key computed by the agents. This threat model is motivated by the use case where a device was fully compromised, except for the long term authentication keys that may be stored securely inside a Trusted Execution Environment, as proposed in [21, p. 48].

The attack corresponds to a Machine-in-the-Middle attack in which the attacker impersonates an initiator I to a responder R, although I did not initiate any session with R. Interestingly, this violation only occurs with methods others than 0. Indeed, even with access to all the ephemeral secrets, the attacker cannot forge a signature. In all other methods, the core issue is the use of the final session key, which is leaked, as the MAC key, allowing the attacker to forge the MAC of the last message.

For readability, we describe in Fig. 2 the attack reported by the tool on a simplified version of EDHOC (omitting message fields that are not relevant). We suppose that I initiates a session with the attacker. The attacker then simply forwards $\mathsf{message}_1$ to R. Then, using I's leaked ephemeral share X and G^Y, the attacker can compute a valid $\mathsf{message}_2$ with their own identity ID_A. The initiator replies with $\mathsf{message}_3$ that would not be accepted by R, as the transcripts do not match. However, after the leak of the session key, the attacker forges a MAC on the expected transcript. R accepts the forged message, and successfully finishes a session supposed to be initiated by I, though I never started a session with R.

This attack highlights an interesting property that a key exchange protocol should guarantee when an implementation with a TEE is considered: entity authentication should only rely on the long term authentication secret of an agent and tolerate compromise of all ephemeral secrets.

Weak data authentication. It is claimed in [21] that an additional key confirmation step implies authentication of all the data, including TH_4. However, the use of the recommended key confirmation is not enforced, and only suggested. Recall that the key material consists in PRK_{4x3m} and TH_4; a natural key confirmation could only rely on PRK_{4x3m} in which case TH_4 is not authenticated.

This behaviour was highlighted by the fact that the authentication of TH_4 could be broken in multiple, implementation dependent, ways, mostly because the ciphertext contained in

Property	**Threat model**					
	Basic	AEAD$^{\lightning}$	DH$^{\lightning}$	DHShare$^{\lightning}$ + SessKey$^{\lightning}$	Hash$^{\lightning}$ + DH$^{\lightning}$	KEM variant
Confidentiality	✓	✓	✓	✓	✗	✓
Implicit& Explicit Key Auth.	✓	✓	✓	✗	✓	✓
Transcript Auth.	✓	✗	✓	✓	✗	✓
Algo Auth.	✓	✓	✓	✓	✗	✓
Session key uniqueness	✓	✓	✗	✓	✗	✗
Non-repudiation soundness	✓	✓	∼	✓	∼	✓
Identity protection	✗	✗	✗	✗	✗	✗

✓: property satisfied ✗: violation of property ∼: unclear security

Table 4: Summary of the automated analysis results over draft 12

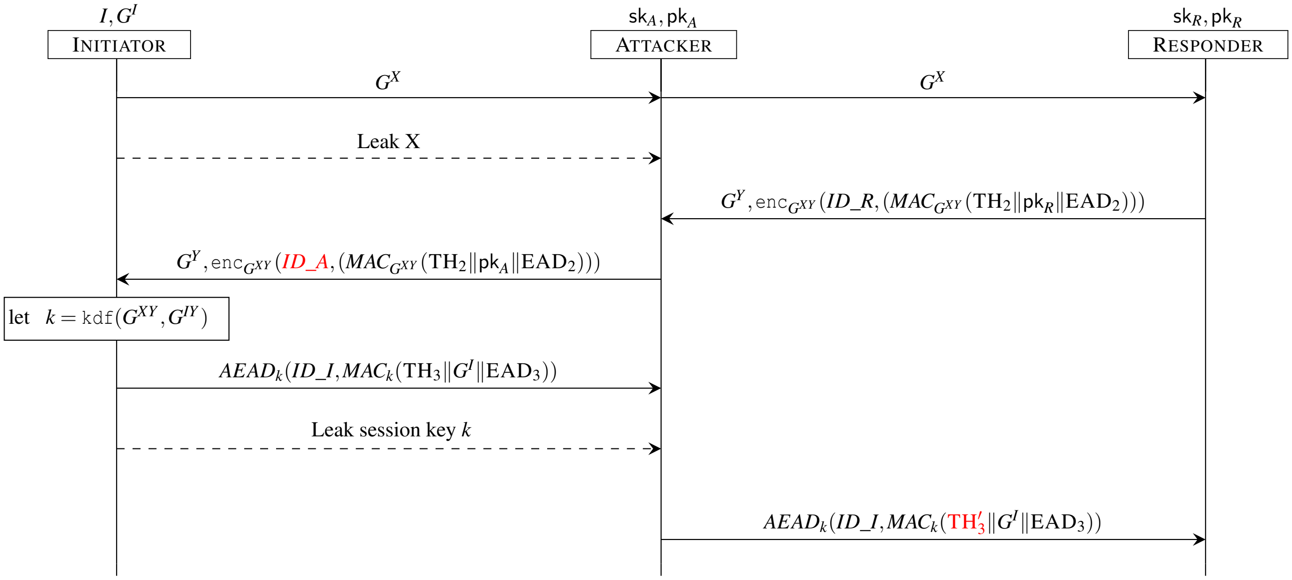
I, G^I
INITIATOR
$\mathsf{sk}_A, \mathsf{pk}_A$
ATTACKER
$\mathsf{sk}_R, \mathsf{pk}_R$
RESPONDER
G^X
G^X
Leak X
$G^Y, \mathtt{enc}_{G^{XY}}(ID\_R, (MAC_{G^{XY}}(\mathrm{TH}_2 \| \mathsf{pk}_R \| \mathrm{EAD}_2)))$
$G^Y, \mathtt{enc}_{G^{XY}}(ID\_A, (MAC_{G^{XY}}(\mathrm{TH}_2 \| \mathsf{pk}_A \| \mathrm{EAD}_2)))$
let $k = \mathtt{kdf}(G^{XY}, G^{IY})$
$AEAD_k(ID\_I, MAC_k(\mathrm{TH}_3 \| G^I \| \mathrm{EAD}_3))$
Leak session key k
$AEAD_k(ID\_I, MAC_k(\mathrm{TH}_3' \| G^I \| \mathrm{EAD}_3))$

Figure 2: Attack authentication in the compromise device but secure TEE setting

$\mathsf{message}_3$ is directly included in the transcript to build TH_4. TH_4 authentication could then be broken when:

- the used AEAD scheme only provides integrity of the plaintext and not of the ciphertext (as it is the case for Mac-then-Encrypt);
- the encoding of $\mathsf{message}_3$ is not fully deterministic;
- when a party recomputes the last message sent from the internal protocol state, as specified in [21, Appendix E, p. 73], but using a randomized signature: this leads to two distinct, but valid versions of $\mathsf{message}_3$.

Out of those three possible behaviours, the first one was reported automatically by the tool in the weak AEAD threat model as an attack on data authentication, which lead us to investigate the other two possibilities.

Session key is weaker than exported keys. In the threat model with low order DH points (Section 3.3), our tool reported that a dishonest party can actually control the key and force it to the neutral DH element. In the KEM variant, a dishonest responder can even fully control the key. While session key uniqueness was not a property claimed by the standard, there was no clear reason why the exported keys would offer this property but not the session key.

We defined and explicitly checked in our analysis for this property, that we called session key uniqueness (this property is also referred to as contributiveness or attacker key control).

Our proposal. We propose to fix all the previous weaknesses by adding a final key derivation as part of the protocol that would depend both on $\mathtt{PRK}_{4x3m}$ and TH_4. Such a session key, intuitively derived as $\mathtt{kdf}(\mathtt{PRK}_{4x3m}, \mathsf{TH}_4)$, has the following benefits:

- the session key differs from the MAC key;
- a key confirmation of the session key implies authentication of all the data;
- a dishonest party cannot control the final value of the session key.

An additional benefit of this proposal is that the final state of the protocol is simplified: only the session key needs to be stored, instead of a key and a transcript. We considered alternatives but preferred the above solution: notably they would increase the complexity of implementations, or solve less issues at once. One option was for instance to enforce key uniqueness by checking that the ephemeral DH shares are not low order points of the curve in use.

4.4 Transcript collision attacks

Using TAMARIN, which has a better support for modeling chosen prefix collision attacks, we found that EDHOC was vulnerable to transcript collision attacks. This is a Machine-in-the-Middle (MitM) attack where the attacker intercepts both ephemeral key shares, and replaces them with the neutral element of the DH group. Then, both parties will have a shared secret equal to this neutral element. Such a MitM attack is generally avoided as parties will not agree on the transcript hash. However, the attacker is able to use the fields (of unbounded length) C_R, C_I or EAD to inject chosen data which allows to create a collision on the hash of the transcripts. Similar attacks were discussed on TLS, IKE and SSH in [5].

Recall that a chosen-prefix collision allows for any p_1, p_2 to compute c_1, c_2 such that $h(p_1||c_1) = h(p_2||c_2)$. We detail the attack below by displaying the *expected* transcript computation T_E, followed by I's transcript T_I and the R's transcript T_R, where we denote by e the neutral DH element and c_1,c_2 the bytes used to obtain a chosen prefixed collision.

$$\begin{array}{llllllll}
T_E := \text{method} & \| \text{ Suites} & \| G^X & \| C_I & \| EAD_1 & \| G^Y & \| C_R & \\
T_I := \quad 0 & \| \text{ Suites}_I & \| G^X & \| C_I & \| EAD_1 & \| \text{ e} & \| \text{c}_1 & \| g^y \| C_R \\
T_R := \quad 0 & \| \text{ Suites}_I & \| \text{ e} & \| C_I & \| \text{ c}_2 & \| G^Y & \| C_R &
\end{array}$$

Due to the chosen prefix collisions, we indeed have that

$$h(0||\text{Suites}_I||G^X||C_I||EAD_1||e||c_1) = h(0||\text{Suites}_I||e||C_I||c_2)$$

and appending to both sides $(G^Y||C_R)$ preserves the collision due to the length extension property. A variant of this attack also allows to perform a downgrade attack. While no hash function currently in the standard does have this weakness, we deemed it important to try to future-proof the protocol against such attacks, especially in the IoT context where it can be difficult to deprecate an algorithm. We proposed several solutions to the WG, which after discussion opted for an alternative which is to change the order of the arguments inside the hash computation. We verified which order would allow to avoid such attacks, and the hash transcript is now computed as:

$$h(G^Y||C_R||h(method||Suites||G^X||C_I||EAD_1))$$

The attack was initially found automatically on the KEM version in 16h using TAMARIN. On the DH version of the protocol we can easily find the attack using TAMARIN's interactive mode.

4.5 Key and IV-reuse

While modeling the protocol and browsing the standard, we manually identified the following weakness. As EDHOC may be deployed on unreliable networks, it may be necessary to re-send the last message.

> **Packet loss resilience** **[21, Appendix E, p. 73]**
>
> *An* EDHOC *implementation MAY keep the protocol state to be able to recreate the previously sent* EDHOC *message and resend it.*

Consider $\mathsf{message}_3$, which is of the form $\mathtt{aead}(SIG, IV, K)$, where IV and K are derived from PRK_{4x3m}. We observe that the computation of the IV and the key K are deterministic w.r.t. to the state of the protocol. However, if a randomized signature scheme is used, such as ECDSA, the plaintext is also randomized. A well-known security issue of AEADs [18, Section 3.1] is that an IV must only be used once for a given ciphertext and key; otherwise no confidentiality nor integrity property may be assumed. Thus, recomputing the message actually leads to a nonce reuse for AEADs, and must not be allowed when using randomized signatures.

Following our recommendation, such behaviour is now forbidden in draft 14 and the security risk is mentioned. Note that we did not try to capture this behaviour in our models of draft 12 as this behaviour completely disappeared in the new version.

4.6 Privacy leak

The initiator's identity should be protected against active attackers that do not own a public key that I is willing to exchange with. However, to mitigate a form of selfie attacks, a party may check that it is not receiving one of its own identities (modeled by Cred-Check). When modeling this particular feature, we observed that an active attacker can then test whether a receiver and an initiator share the same identity by observing the failure of the exchange due to the aforementioned test. In turn, as an active attacker can learn the identity of any responder, it can also learn the identity of any initiator.

It was clarified with the WG that this mitigation was intended for the Trust On First Use setting, in which anybody can learn the identity of initiators, as they are willing to exchange with anybody. However, this lead to the observation

Attack type	Requirements	Found by	Action
Initiator Impersonation	Ephemeral share and Session key leaks	PROVERIF (846 s)	✓(draft 14)
Secrecy & Auth. breach & Downgrade attack	Hash Chosen-prefix collisions and no neutral DH check	TAMARIN (16 h)	✓(draft 14)
Final transcript mismatch	Leak session key or Non deterministic encoding or Leak share and Malleable Sig.	PROVERIF (56 s)	✓(draft 14)
Party Controlled Session key	No neutral DH check or KEM variant	PROVERIF (49 s)	✓(draft 14)
Identity leak	Initiator refuses to exchange with its identity	DEEPSEC (1 s)	To be clarified
Duplicated non-repudiation	Malleable Sig.	PROVERIF (81 s)	Judged irrelevant
AEAD Key/IV reuse	Message recomputation from stored state	Manual	✓(draft 14)

Table 5: Summary of our attacks and action takens

that in the classical setting, where each party has a list of trusted identities, an active attacker can test whether an identity is allowed or not. This leads to a privacy leak when the list of trusted identities for I is only missing I's identity.

Details regarding the verification results for anonymity are given in Appendix B. As our privacy study relies on DEEPSEC which does not support the DH exponentiation, our results are only for the KEM based method. In ProVerif we additionally studied the DH version, but only for method 0 because of scalability issues.

4.7 Non-repudiation

We specified non-repudiation (soundness) as an injective property. This models that one proof presented to the judge should correspond to a single session. We identified several weaknesses that could make the non-repudiation guarantees unclear. Notably, it could be tricky to implement a judge that would need to count the number of sessions performed by a party because:

- first, if the parties accept low-order points or the identity group element, many sessions may share the same PRK_{4x3m};
- second, if the signature is malleable (i.e., given one signature for a message, it is possible to create a second distinct signature for the same message), multiple different proofs can be produced for the same session.

In addition, we witnessed that a dishonest agent could make itself repudiable by choosing a weak signature key for which verification will succeed over any message for some signature schemes and notably ED25519. After discussions, the WG considered that such considerations were side cases that did not need to be detailed in the draft.

4.8 Analysis of the latest draft

Overall, we introduced in draft 14 a major change in the key derivation by adding a final key, we updated the order of the transcript elements, and we avoid an IV and key reuse over an AEAD. We performed again the analysis of the most significant scenarios with PROVERIF, confirming that previous weaknesses are mitigated by our changes. In addition, we used TAMARIN with a long time-out of 24 hours and 8 cores per job to try to obtain stronger guarantees over some scenarios. While many scenarios produced time-outs, we were able to strengthen the guarantees for 13 advanced threat models for an accumulated TAMARIN runtime of 82 hours.

5 Conclusion

In this paper, we illustrate how recent state-of-the-art developments from the formal method community can significantly leveraged in the standardization processes of security protocols. Concretely, we performed an automated analysis of the EDHOC protocol (draft 12). Building on automated tools and a systematized workflow allowed us to consider combinations of strong threat models and identify multiple ways to strengthen the protocol. Those were reported to the WG and lead to lively interactions over GitHub issues and pull-requests. Several of our findings lead to changes integrated in draft 14 which we subsequently verified to confirm that the proposed changes mitigate previously identified weaknesses.

In the future, we aim at maintaining our models up to date with upcoming drafts until the final version of the standard, in order to provide a reference model, in the spirit of a reference implementation. The process of updating the models should greatly benefit from the modular architecture that we set up. In addition, there are multiple small ways to improve our models notably by adding the optional message four or the integrated session key update mechanism, but that may come at the cost of a loss of automation and a reduction of the scope of threat models considered.

Acknowledgments This work has been partly supported by the ANR Research and teaching chair in AI ASAP (ANR-20-CHIA-0024) and ANR France 2030 project SVP (ANR-22-PECY-0006).

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1), 2018.

[2] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *Conference on Computer and Communications Security*. ACM, 2018.

[3] David A. Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.*, 17(2), 2014.

[4] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[5] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH. In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2016.

[6] Bruno Blanchet, Vincent Cheval, and Cortier Véronique. Proverif with lemmas, induction, fast subsumption, and much more. In *Proceedings of the 43th IEEE Symposium on Security and Privacy (S&P'22)*. IEEE Computer Society Press, May 2022.

[7] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. Hash gone bad: Automated discovery of protocol attacks that exploit hash function weaknesses. In *32st USENIX Security Symposium*. USENIX Association, 2023.

[8] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. SAPIC^{+}: protocol verifiers of the world, unite! In *31st USENIX Security Symposium*. USENIX Association, 2022.

[9] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018.

[10] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of tls 1.3. In *Conference on Computer and Communications Security (CCS'17)*. ACM, 2017.

[11] Cas Cremers and Dennis Jackson. Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using diffie-hellman. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019.

[12] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 1981.

[13] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. A spectral analysis of noise: A comprehensive, automated, formal analysis of Diffie-Hellman protocols. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, August 2020.

[14] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019.

[15] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. Lake edhoc models. `https://github.com/charlie-j/edhoc-formal-analysis`.

[16] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. *ACM Trans. Priv. Secur.*, 24(2), jan 2021.

[17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European symposium on security and privacy (EuroS&P 2017)*. IEEE, 2017.

[18] David McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116, January 2008.

[19] Karl Norrman, Vaishnavi Sundararajan, and Alessandro Bruni. Formal analysis of EDHOC key establishment for constrained iot devices. *CoRR*, abs/2007.11427, 2020.

[20] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *25th Computer Security Foundations Symposium (CSF 2012)*. IEEE, June 2012.

[21] Göran Selander, John Preuß Mattsson, and Francesca Palombini. Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-12, Internet Engineering Task Force, October 2021. Work in Progress.

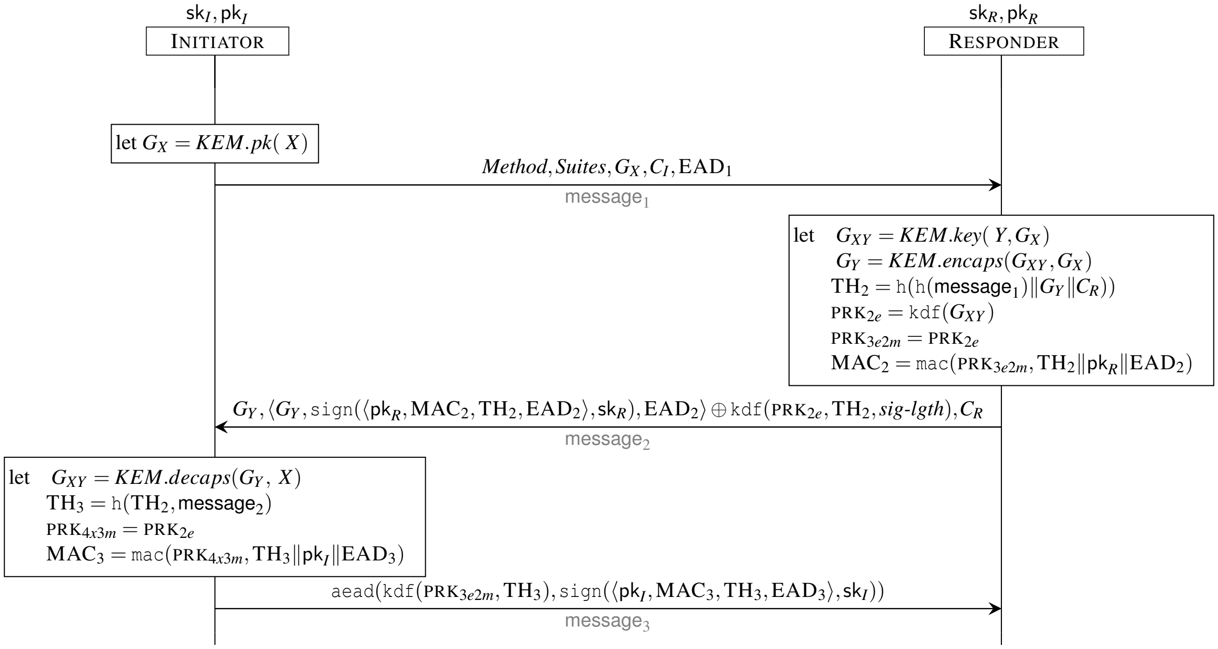
$\mathsf{sk}_I, \mathsf{pk}_I$
INITIATOR
$\mathsf{sk}_R, \mathsf{pk}_R$
RESPONDER
let $G_X = KEM.pk(\ X)$
$Method, Suites, G_X, C_I, \mathrm{EAD}_1$
$\mathsf{message}_1$
let $G_{XY} = KEM.key(\ Y, G_X)$
$G_Y = KEM.encaps(G_{XY}, G_X)$
$\mathrm{TH}_2 = \mathtt{h}(\mathtt{h}(\mathsf{message}_1) \| G_Y \| C_R))$
$\mathrm{PRK}_{2e} = \mathtt{kdf}(G_{XY})$
$\mathrm{PRK}_{3e2m} = \mathrm{PRK}_{2e}$
$\mathrm{MAC}_2 = \mathtt{mac}(\mathrm{PRK}_{3e2m}, \mathrm{TH}_2 \| \mathsf{pk}_R \| \mathrm{EAD}_2)$
$G_Y, \langle G_Y, \mathtt{sign}(\langle \mathsf{pk}_R, \mathrm{MAC}_2, \mathrm{TH}_2, \mathrm{EAD}_2 \rangle, \mathsf{sk}_R), \mathrm{EAD}_2 \rangle \oplus \mathtt{kdf}(\mathrm{PRK}_{2e}, \mathrm{TH}_2, sig\text{-}lgth), C_R$
$\mathsf{message}_2$
let $G_{XY} = KEM.decaps(G_Y,\ X)$
$\mathrm{TH}_3 = \mathtt{h}(\mathrm{TH}_2, \mathsf{message}_2)$
$\mathrm{PRK}_{4x3m} = \mathrm{PRK}_{2e}$
$\mathrm{MAC}_3 = \mathtt{mac}(\mathrm{PRK}_{4x3m}, \mathrm{TH}_3 \| \mathsf{pk}_I \| \mathrm{EAD}_3)$
$\mathtt{aead}(\mathtt{kdf}(\mathrm{PRK}_{3e2m}, \mathrm{TH}_3), \mathtt{sign}(\langle \mathsf{pk}_I, \mathrm{MAC}_3, \mathrm{TH}_3, \mathrm{EAD}_3 \rangle, \mathsf{sk}_I))$
$\mathsf{message}_3$

Figure 3: EDHOC protocol, KEM method

[22] Marc Stevens. *A Survey of Chosen-Prefix Collision Attacks*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021.

[23] Mališa Vučinić, Göran Selander, John Preuss Mattsson, and Thomas Watteyne. Lightweight authenticated key exchange with EDHOC. *Computer*, 55(4), 2022.

A The KEM-based version of EDHOC

A detailed presentation of the KEM-based version of EDHOC is depicted in Figure 3. This variant is rather similar to method 0 but replaces the DH key exchange by a KEM based one.

B Detailed analysis results

The following tables (Tables 7 and 8) present the detailed results of our analysis, along with the required verification times. The results are presented per lemma (expressing the security properties, as explained in Section 3.2). The four others columns correspond to the protocol version that is analysed. The results in Draft 12 and Draft 12 KEM correspond to the version of the draft we originally analysed while the results in the Draft 14 and Draft 14 KEM correspond to the new versions of the protocol including mitigations we suggested.

For each lemma, the results are presented as follow:

- The scenario: a summary of the attacker capabilities and optional checks that are considered in this specific run. The explanation for the abreviations used can be found in Table 3 and at the bottom of each table;
- The result of the verification: it can be an attack (✗) or a proof (using PROVERIF: $✓^P$, or using TAMARIN: $✓^T$);
- The verification time.

Due to lack of space, we omit the results for secretRwhich are similar to secretI, for auth-RI-uniquewhich are similar to auth-IR-unique, as well as no-reflection-attacks-RI which instantly holds when we add the Cred-Check. The full and latest results over the draft can be found in [15].

The results on anonymity are presented in a separated table (Table 6) indicating the precise protocol, whether the Cred-Check test is performed and whether we test the property for a bounded or unbounded number of sessions. $✓^D$ and $✗^D$ indicate that the DEEPSEC tool was used.

	Draft 12	**Draft 14**
KEM w/ Cred-Check , bounded sessions	$✗^D$ (3s)	$✗^D$ (3s)
KEM w/o Cred-Check , bounded sessions	$✓^D$ (3s)	$✓^D$ (3s)
KEM w/o Cred-Check, unbounded sessions	$✓^P$ (10m)	$✓^P$ (6m)
DH w/o Cred-Check, unbounded sessions, only method 0	$✓^P$ (267m)	$✓^P$ (148m)

Table 6: Summary of our results for anonymity

Lemma	Draft 12	Draft 12 KEM	Draft 14	Draft 14 KEM
secretI	DHShare$^{\lightning}$, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (1076s)	DHShare$^{\lightning}$, SessKey$^{\lightning}$, $\oplus^{\lightning}$, AEAD$^{\lightning}$, Sig$^{\lightning}$-proof ✓^{P} (316s)	DHShare$^{\lightning}$, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (1833s)	Sig$^{\lightning}$-proof, DHShare$^{\lightning}$, SessKey$^{\lightning}$, $\oplus^{\lightning}$, AEAD$^{\lightning}$ ✓^{P} (259s)
	$\oplus^{\lightning}$ ✓^{P} (104s)		$\oplus^{\lightning}$ ✓^{P} (278s)	
	AEAD$^{\lightning}$, Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (68s)		AEAD$^{\lightning}$, Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (143s)	
	SessKey$^{\lightning}$, Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (78s)		SessKey$^{\lightning}$, Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (190s)	
			- ✓^{T} (426m)	
auth-IR-unique	- ✗ (49s)	- ✗ (12s)	- ✓^{P} (51s) ✓^{T} (30m)	- ✓^{P} (11s)
	DH-Check, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (98s)		DH-Check, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (67s) ✓^{T} (30m)	
	DH-Check, AEAD$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (48s)		DH-Check, AEAD$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (52s) ✓^{T} (114m)	
	DH-Check, $\oplus^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (104s)		DH-Check, $\oplus^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (107s)	
data-authentication-IR	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (116s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✓^{P} (20s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (125s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✓^{P} (21s)
	$\oplus^{\lightning}$ ✓^{P} (73s)	$\oplus^{\lightning}$ ✓^{P} (13s)	$\oplus^{\lightning}$ ✓^{P} (75s)	$\oplus^{\lightning}$ ✓^{P} (14s)
	SessKey$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (66s)	SessKey$^{\lightning}$, AEAD$^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (12s)	SessKey$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (89s) ✓^{T} (1110m)	SessKey$^{\lightning}$, AEAD$^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (14s)
	AEAD$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (65s)		AEAD$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (64s)	

Table 7: Detailed results
Attackers capabilities: Sig$^{\lightning}$: precise signatures (for attack finding), Sig$^{\lightning}$-proof: precise signature (stronger guarantees), DH$^{\lightning}$: precise DH with small subrougps, AEAD$^{\lightning}$: precise AEAD model, $\oplus^{\lightning}$: malleable xor encryption, SessKey$^{\lightning}$: compromise of session keys, DHShare$^{\lightning}$: compromise ephemeral shares
Protocol optional checks: DH-Check: neutral DH element check, Cred-Check: own identity check

Lemma	Draft 12	Draft 12 KEM	Draft 14	Draft 14 KEM
data-authentication-RI	AEAD$^{\lightning}$ ✗ (56s)	AEAD$^{\lightning}$ ✗ (13s)	AEAD$^{\lightning}$ ✓^{P} (60s) ✓^{T} (546m)	AEAD$^{\lightning}$ ✓^{P} (14s)
	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (863s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (27s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (892s)	DHShare$^{\lightning}$, Sig$^{\lightning}$ ✗ (28s)
	SessKey$^{\lightning}$, Sig$^{\lightning}$ ✗ (180s)	SessKey$^{\lightning}$ Sig$^{\lightning}$-proof ✗ (15s)	SessKey$^{\lightning}$, Sig$^{\lightning}$ ✓^{P} (90s) ✓^{T} (343m)	SessKey$^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (15s)
	$\oplus^{\lightning}$ ✓^{P} (160s)	$\oplus^{\lightning}$ ✓^{P} (17s)	$\oplus^{\lightning}$ ✓^{P} (187s)	$\oplus^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (30s)
honest-auth-RI-non-inj	DHShare$^{\lightning}$, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✗ (846s)	DHShare$^{\lightning}$, AEAD$^{\lightning}$, $\oplus^{\lightning}$, SessKey$^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (504s)	DHShare$^{\lightning}$, Sig$^{\lightning}$, SessKey$^{\lightning}$ ✓^{P} (780s) ✓^{T} (391m)	DHShare$^{\lightning}$, AEAD$^{\lightning}$, $\oplus^{\lightning}$, SessKey$^{\lightning}$ Sig$^{\lightning}$-proof ✓^{P} (69s)
	SessKey$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (91s)		SessKey$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (108s) ✓^{T} (846m)	
	AEAD$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (84s)		AEAD$^{\lightning}$ Sig$^{\lightning}$-proof, DH$^{\lightning}$ ✓^{P} (90s)	
	$\oplus^{\lightning}$ ✓^{P} (104s)		$\oplus^{\lightning}$ ✓^{P} (139s)	
repudiation-soundness	- ✗ (81s)	- ✗ (103s)	- ✓^{P} (34s)	- ✓^{P} (15s)
	DH-Check, Sig$^{\lightning}$ ✗ (125s)		DH-Check, Sig$^{\lightning}$ ✗ (36s)	
	DH-Check ✗ (74s)		DH-Check ✓^{P} (25s)	
	DH-Check, AEAD$^{\lightning}$ ✗ (86s)		DH-Check, AEAD$^{\lightning}$ ✓^{P} (28s)	

Table 8: Detailed results
Attackers capabilities: Sig$^{\lightning}$: precise signatures (for attack finding), Sig$^{\lightning}$-proof: precise signature (stronger guarantees), DH$^{\lightning}$: precise DH with small subrougps, AEAD$^{\lightning}$: precise AEAD model, $\oplus^{\lightning}$: malleable xor encryption, SessKey$^{\lightning}$: compromise of session keys, DHShare$^{\lightning}$: compromise ephemeral shares
Protocol optional checks: DH-Check: neutral DH element check, Cred-Check: own identity check