Extended Formal Analysis of the EDHOC Protocol in Tamarin

Karl Norrman[1,2][0000−0003−0164−1478], Vaishnavi Sundararajan[3][0000−0002−5945−5208], and Alessandro Bruni[4]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
[2] Ericsson Research, Security, Stockholm, Sweden
karl.norrman@ericsson.com
[3] University of California, Santa Cruz, USA
vasundar@ucsc.edu
[4] IT University of Copenhagen, Copenhagen, Denmark
brun@itu.dk

Abstract. Given how common IoT devices that use constrained resources are becoming today, the need of the hour is communication protocols which can operate securely under such limitations. For a few years, the Internet Engineering Task Force (IETF) has been working to standardize EDHOC, an authenticated key establishment protocol for such constrained IoT devices. The first version of EDHOC was proposed in 2016. In 2018, Bruni et al. [Bruni et al., 2018] used the ProVerif tool [Blanchet, 2001] to formally analyze an early version of EDHOC, which had only two key establishment methods. By 2021, the protocol had been fleshed out much more, with multiple new key establishment methods, and this version was formally analyzed using the Tamarin prover [Meier et al., 2013] in [Norrman et al., 2021]. In this paper, we build on that work, by modifying the model, analyzing some new properties, and discussing some aspects of the latest EDHOC specification. In particular, we extend the modeling in [Norrman et al., 2021] with trusted execution environments (TEEs), modify the way we model XOR encryption, and in addition to the properties verified in [Norrman et al., 2021], we verify weak post-compromise security (PCS) as well as the secrecy and integrity of some additional data used as part of the protocol.

Keywords: Formal Verification, Symbolic Dolev-Yao Model, Authenticated Key Establishment, Protocols, IoT, Tamarin.

1 INTRODUCTION

IoT protocols are often run on devices which operate under severe restrictions on resources like bandwidth and energy consumption. These constrained devices are often simple in their operation, but need to communicate and function without human interference or maintenance for extended periods of time. The Internet Engineering Task Force (IETF) is standardizing new protocols to secure communications between devices that operate under such restrictions. One such is the Object Security for Constrained RESTful Environments (OSCORE) protocol. However, OSCORE requires the pre-establishment of a security context. To this end, a key exchange protocol named

Ephemeral Diffie-Hellman Over COSE (EDHOC) is being discussed in the IETF. Since EDHOC will establish security contexts for OSCORE, the same resource constraints (especially those pertaining to message size) apply to the former as for the latter. While establishing security contexts for OSCORE is the primary goal for the EDHOC protocol, there might well be other use cases which have not been explored in depth yet. It is therefore important to ensure that fundamental properties expected of key exchange protocols as established in the literature are satisfied by EDHOC as well.

1.1 Evolution of EDHOC

The first EDHOC framework was introduced in March 2016. It allowed two different key establishment methods – one involved a pre-shared Diffie-Hellman (DH) *cryptographic core*, and the other was a variation on challenge-response signatures, à la OPTLS [Krawczyk and Wee, 2016].

A *cryptographic core*, often just called a core, is an academic protocol, i.e., with no encodings or application-specific details as needed for an industrial protocol. Once these ingredients are added to a cryptographic core, we obtain a key-establishment method. Since then, the protocol has seen multiple changes. In May 2018, the designers replaced the challenge-response signature core with one based on SIGMA (SIGn-and-MAc) [Krawczyk, 2003,Selander et al., 2018], and in 2020, three new cores, which mixed challenge-response signatures and regular signatures were added as well [Selander et al., 2020]. While this is the version we base our formal analysis on, there have been later versions. See Section 7.1 for a more detailed discussion.

1.2 Related Work and Contributions

The earliest related work is [Bruni et al., 2018], which formally analyzes the May 2018 version of EDHOC using the ProVerif tool [Blanchet, 2001]. In this paper, the authors analyze two key establishment methods – one built on a pre-shared key authentication core, and one based on SIGMA. The authors check the following properties: secrecy, identity protection, strong authentication, perfect forward secrecy (PFS), and integrity of application data. Later work [Norrman et al., 2021] analyzes the July 2020 version of EDHOC in the Tamarin prover [Meier et al., 2013]. This version of the protocol has four key establishment methods. In [Norrman et al., 2021], the properties checked for are injective agreement, implicit agreement, and perfect forward secrecy for the session key material. That work also includes a discussion about the various design choices made as part of EDHOC, and the impact of EDHOC in multiple use-case scenarios.

1.3 Contributions

In this paper, we extend the work presented in [Norrman et al., 2021]. We formally analyze the EDHOC specification as of July 2020 [Selander et al., 2020], but our formal analysis applies as far as the version of the specification from February 2021. We extend the adversary model and Tamarin system models to capture weak post-compromise security (PCS) and model Trusted Execution Environments (TEE). We also alter the formal modeling used for encryption under XOR. We formally verify the following properties:

- Injective agreement
- Implicit agreement
- Perfect Forward Secrecy (PFS) for the session-key material
- Weak post-compromise security for the session-key material
- Secrecy and integrity of ad_3

We follow the definition of weak PCS by [Cohn-Gordon et al., 2016], which subsumes PFS. Since EDHOC is still a protocol under development, the latest version of EDHOC is the one from July 2022 [Selander et al., 2022], which contains details which are not covered by our formal model. We do, however, refer to various aspects (error handling, denial of service etc) of the July 2022 version of the specification, namely in Sections 5 and 6. We also discuss various issues arising due to the use of trusted execution environments (TEEs), denial of service (DoS) attacks, error handling, the negotiation of parameters (which the formal model abstracts away) for establishing the protocol, and other potential attacks and concerns. We have communicated these issues to the developers of the protocol.

2 THE EDHOC PROTOCOL

In this section, we describe the various key establishment methods of the EDHOC protocol. Following [Norrman et al., 2021], we refer to the two roles executing the protocol as the initiator I and the responder R. We annotate values with I and R to make explicit which role they belong to.

2.1 Notation

We denote by $\langle d_{t,id}, Q_{t,id} \rangle$ public-private key pairs, where d is the private key, Q is the public key, $t \in \{e, s\}$ indicates whether the key is ephemeral or static, and id indicates the party to whom the key pair belongs. When clear from context, we will often drop some (or all) of the subscripts. Static key pairs (suitable for regular or challenge-response signatures) are long-term authentication credentials, whereas ephemeral key pairs are those generated afresh for each execution of the protocol.

Parties can authenticate using regular signatures or challenge-response signatures. In the former case, we say that they use the *signature-based authentication method* (SIG). In the latter case, we say, following the terminology in the specification, that they use the *static key authentication method* (STAT). We adopt the challenge-response terminology from [Krawczyk, 2005].

EDHOC fundamentally uses elliptic curves and associated Diffie-Hellman operations. Signatures using a party A's keys are denoted by $sign_A(\cdot)$, while the verification thereof is denoted by $vf_A(\cdot)$. A Diffie-Hellman operation which combines a private key d and a point P on the elliptic curve is represented as $dh(d, P)$. We will often overload notation to let P stand for both the point on the elliptic curve as well as the corresponding bitstring encoding.

2.2 Overall Description

EDHOC is designed to establish a security context for the OSCORE protocol. This context, in particular, includes the session-key material (we denote this by Z). The generalized abstract protocol as given in the specification [Selander et al., 2020] consists of three messages, and is shown in Figure 1. The abstract structure is the same across methods. However, the authentication mechanisms and key derivation procedures differ between methods. EDHOC may also transfer application data ad_1, ad_2, and ad_3 in addition to establishing the OSCORE security context.

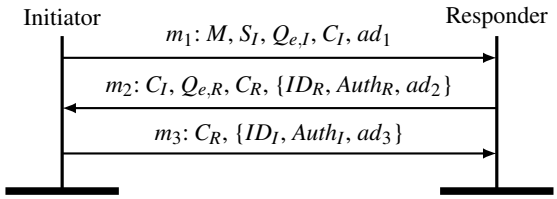

Fig. 1. Structure of EDHOC: $\{t\}$ means t is encrypted and integrity protected. [Norrman et al., 2021]

Of the three messages m_1, m_2, and m_3, the first two, among other things, establish a common authentication method M and ciphersuite S_I. The party playing the initiator role uses M to propose which authentication methods the two parties shall use, and in S_I, proposes an ordered list of choices for the ciphersuite. The chosen authentication methods may differ for the two roles, yielding four possible combinations: SIG-SIG, SIG-STAT, STAT-SIG, and STAT-STAT, where the first authentication method in each combination is used by the initiator, and the second by the responder. [5] The party executing the responder role may choose to reject the method and/or ciphersuite chosen by the initiator by sending an error message. This results in abandoning this session and renegotiating, as the initiator goes down their list of choices for ciphersuites, and picks the next option for a next execution of the protocol. Our analysis does not cover such renegotiation which requires maintaining state between executions to remember the rejected ciphersuites. However, we will discuss the ramifications of such a renegotiation procedure and the error messages later, in Section 5.

In addition to negotiating the method and ciphersuite, the first two messages are also instrumental for the exchange of public ephemeral keys $Q_{e,I}$ and $Q_{e,R}$, and connection identifiers C_I and C_R, for the initiator and responder roles respectively. The specification states that the connection identifiers serve only to route messages to the correct party executing EDHOC, but also claims that they may be used in turn by protocols (like OSCORE) which use the security context established by EDHOC. While the specification does not require any explicit security guarantees to be satisfied by these connection

[5] As in the specification, we will from now on overload notation and refer to the combinations of authentication methods as methods as well.

identifiers, it does, however, require that the identifiers be unique, i.e., in any session, $C_I \neq C_R$, and that the parties involved in the session can verify this uniqueness. More precisely, the specification states that OSCORE should be able to use these identifiers to retrieve any particular security context. In this work, as in [Norrman et al., 2021], we verify that the parties agree on the values of C_I and C_R. The second and third messages also serve to identify and authenticate each party to the other. These messages contain long-term key identifiers (ID_I and ID_R). Additionally, the messages contain authenticating information ($Auth_I$ and $Auth_R$), which lets each party know that the other party does indeed control the long-term key associated with these identifiers. The authentication information is structured differently for each authentication method.

Consider the following scenario. The initiator I chooses the method SIG-STAT, and sends this via M to the responder R. R now has the option to reject this choice of method. However, if R chooses to accept this method, they need to provide an identifier ID_R for a key pair which can be used with challenge-response signatures as well as authenticating information $Auth_R$, as dictated by the static key authentication method STAT. I will then respond with an identifier ID_I corresponding to a key pair, which is suitable for regular signatures, and provide authentication $Auth_I$ as appropriate for the signature-based method SIG.

2.3 Key Schedule

The second and third messages of EDHOC contain authenticating information. This information is either a signature or a message authentication code (MAC), as we will describe in more detail in Section 2.4. The keys for these are generated using a key schedule which is intrinsic to the functioning of EDHOC. The key schedule takes a DH key P_e as basic input and builds upon it, as shown in Figure 2.

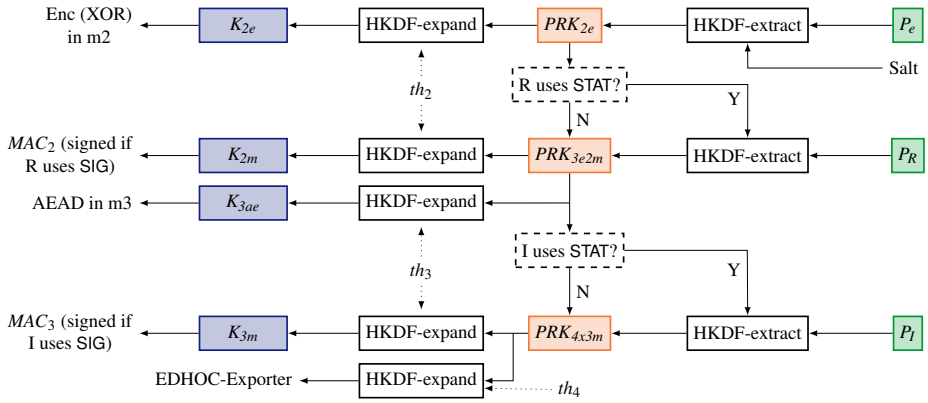

Fig. 2. Key schedule for [Selander et al., 2020]: P_e, P_I, P_R are the DH keys, PRK_{2e}, PRK_{3e2m}, PRK_{4x3m} are the intermediate key material, and K_{2e},K_{2m}, K_{3ae}, K_{3m} are the encryption keys for AEAD or XOR. Dashed boxes are conditionals. [Norrman et al., 2021]

To derive keys, EDHOC uses two functions from the HKDF interface [Krawczyk and Eronen, 2010], HKDF-extract and HKDF-expand. Both functions take as argument two values – an input and a salt. For HKDF-extract, the input is a DH key, while for HKDF-expand, it is intermediate key material.

As mentioned earlier, the fundamental building block for the key schedule is the ephemeral DH key P_e, which is computed in two different ways by I (as $dh(d_{e,I}, Q_{e,R})$) and R (as $dh(d_{e,R}, Q_{e,I})$). This key gives rise to intermediate keys PRK_{2e}, PRK_{3e2m} and PRK_{4x3m}, which can be derived as part of protocol execution. Each intermediate key gives rise to encryption and integrity keys (K_{2e}, K_{2m}, K_{3ae}, and K_{3m}) corresponding to each message in the protocol.

In order to generate the final keys, the two HKDF algorithms use various values for salt. PRK_{2e} is generated by the HKDF-extract algorithm while using the empty string as the salt. PRK_{3e2m} and PRK_{4x3m} are separately generated if R or I uses the STAT method, using the corresponding DH key as input and the previous intermediate key as salt. The key P_R, which is computed as $dh(d_{e,I},\ Q_{s,R}) = dh(d_{s,R},\ Q_{e,I})$, is used if the responder uses the STAT authentication method. Similarly, the key P_I is used if the initiator uses STAT, and is computed as $dh(d_{e,R},\ Q_{s,I}) = dh(d_{s,I},\ Q_{e,R})$.

These intermediate keys are fed into HKDF-expand, which uses as salt a value th, which is a running hash of the information transmitted thus far as part of the protocol. By th_i, we denote the hash corresponding to the i^{th} message.

At the end of a successful run of the protocol, the session-key material is established as Z, which we define as a set of various keys. This set always includes P_e, and if the initiator (resp. the responder) uses the STAT authentication method, then it also includes P_I (resp. P_R). We discuss which material should be included in Z in more detail in Section 4, and the consequences of various choices in Section 6.3. Once Z has been established, an HKDF-based key exporter named EDHOC-Exporter extracts the keys required by the security protocol.

2.4 About Authentication in EDHOC

We now describe how the authentication information is constructed, depending on whether the SIG or STAT method is used. For both methods, the following information is used to compute $Auth_R$: ID_R, $Q_{s,R}$, a transcript hash of all the communicated information thus far in the protocol, and ad_2, if included. $Auth_I$ uses the same pieces of information, but corresponding to the initiator role. A MAC is obtained by feeding this material as additional data and the empty string as input to the Authenticated Encryption with Additional Data (AEAD) algorithm as indicated in the established ciphersuite S_I. The encryption key for the AEAD algorithm is constructed, for both roles, using the ephemeral key material $Q_{e,I}$, $Q_{e,R}$, $d_{e,I}$, and $d_{e,R}$. The initiator computes $dh(d_{e,I}, Q_{e,R})$ while the responder computes $dh(d_{e,R}, Q_{e,I})$, and DH operations give rise to the same key under both these computations.

When I uses the SIG authentication method, $Auth_I$ is I's signature over the MAC computed as above, along with the data covered by the MAC. However, when I uses the STAT method, $Auth_I$ is just the MAC, with one difference: the MAC key is derived using both the ephemeral key material P_e as well as the long-term key for the initiator $\langle d_{s,I},\ Q_{s,I}\rangle$. This is similar to the 1-RTT semi-static pattern in OPTLS which computes

the MAC key sfk for the message sfin [Krawczyk and Wee, 2016]. The same procedures works for R as well (with the values corresponding to R). The abstract calculation for the authentication information values is as shown in Table 1 [Norrman et al., 2021]. We use MAC_R (resp. MAC_I) to denote a MAC which uses an encryption key constructed using $\langle d_{s,R}, Q_{s,R} \rangle$ and $\langle d_{e,I}, Q_{e,I} \rangle$ (resp. using $\langle d_{s,I}, Q_{s,I} \rangle$ and $\langle d_{e,R}, Q_{e,R} \rangle$).

M	$Auth_I$	$Auth_R$
SIG-SIG	$sign_I(\cdot)$	$sign_R(\cdot)$
SIG-STAT	$sign_I(\cdot)$	$MAC_R(\cdot)$
STAT-SIG	$MAC_I(\cdot)$	$sign_R(\cdot)$
STAT-STAT	$MAC_I(\cdot)$	$MAC_R(\cdot)$

Table 1. The outer functions for each method M [Norrman et al., 2021]

In addition to this, the second and third messages have provisions for transferring application messages which are encrypted and integrity protected, as can be seen in 1. The second message is encrypted by performing a bit-wise XOR between the plaintext and the output of the key derivation function HKDF, as in Section 2.3. For the third message, encryption and integrity protection is assured by the AEAD algorithm.

In Figure 3 we show an example of protocol execution under the SIG-STAT method. The figure describes in detail the various message patterns, operations and key derivations used to construct these messages.

3 Implementation Aspects and Key Protection

Authentication of a specific IoT device assumes that the device is the only entity with access to the long-term key associated with its identity. Since IoT devices may be accessible to adversaries, e.g., an insider cloning a key card, long-term keys must be appropriately protected. A state-of-the-art approach is to use a trusted execution environment (TEE), which holds the key and provides an API for operations using the key. This is the approach taken by the TrustZone-based μEDHOC [Hristozov et al., 2021] implementation, for example. Typical operations include signatures using the long-term private key of a party.

TEEs can be of differing complexity. Some, like ARM TrustZone and Intel SGX, are general-purpose execution environments, which can be programmed in many different ways. Others, like the Universal Subscriber Identity Modules (USIM) used for authentication to 3GPP mobile networks, have application-specific interfaces for authentication, key agreement protocols etc.

A fundamental aspect of TEEs is how much of the application is placed in the TEE. For larger devices that include general-purpose processors with TrustZone or SGX, entire EDHOC and OSCORE implementations may reside inside the TEE. For constrained IoT devices on the lower end of the scale, a TEE might have to be implemented using a special-purpose integrated circuit. In the latter case, it may be beneficial to follow a minimalistic approach, and only store in the TEE the long-term key and operations that need access to it, to reduce cost.

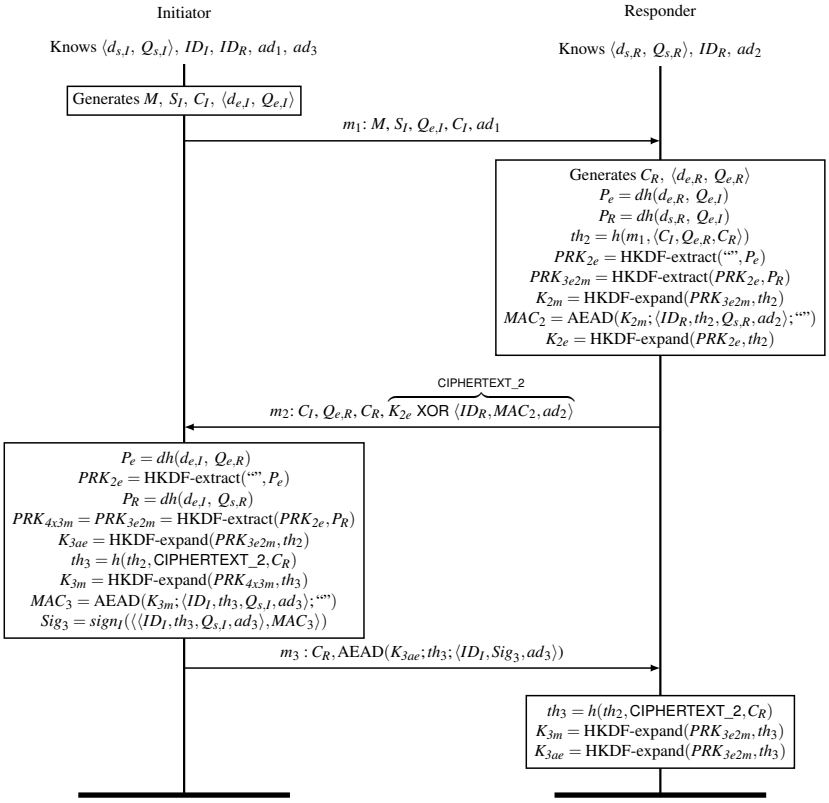

Fig. 3. The SIG-STAT method for EDHOC. $\langle\cdot\rangle$ denotes a tuple, and the hash function h is as established in the ciphersuite S_I. [Norrman et al., 2021]

In general, it might appear more secure to implement as much as possible inside the TEE, but there is a security trade-off. Because security-critical code runs in the same area as where long-term keys reside, an implementation error here risks leaking information of the key to the adversary. From this perspective, it might be beneficial to follow the minimalistic approach even when having access to TrustZone or SGX.

A slightly more secure division of functionality is to keep both the long-term key and the session key inside the TEE and extend the interface to accept messages and return the (en/de)crypted counterpart, i.e. the interface exposes AEAD functions.

4 FORMALIZATION AND RESULTS

The EDHOC specification [Selander et al., 2020] claims that EDHOC satisfies many security properties, but these are imprecisely expressed and motivated. In particular, there is no coherent adversary model. It is therefore not clear in which context properties should be verified. We resolve this by clearly specifying an adversary model, in which we can verify properties.

4.1 Formal Model

As in [Norrman et al., 2021], we verify EDHOC in an extended Dolev-Yao model [Dolev and Yao, 1983]. The Dolev-Yao model is well established for the symbolic verification of security protocols. Messages are modelled as terms in an algebra, and the various cryptographic operations are assumed to be perfect, e.g., encrypted messages can only be decrypted with the correct key, there are no hash collisions etc.

The adversary is assumed to be in control of the communication channel and can see all messages being communicated as part of the protocol. In addition, they can interact with an unbounded number of protocol sessions, and drop, inject and modify messages at will.

On top of the standard Dolev-Yao model, [Norrman et al., 2021] allows the adversary to access long-term and ephemeral keys via specific events. Long-term key reveal, denoted by $\mathbf{A}^t_{\mathsf{LTK}}(A)$, stands for the adversary gaining access to a party A's long-term private key $d_{s,A}$ at a time point t. Ephemeral key reveal, denoted by $\mathbf{A}^t_{\mathsf{Eph}}(A,k)$, stands for the adversary obtaining, at time t, the ephemeral private key $d_{e,A}$ used by party A in a session where they establish a session key k. Formalizing these two capabilities allows more fine-grained control over the access that an adversary has to these fundamentally different kinds of keys. Furthermore, we modify the model from [Norrman et al., 2021] by strengthening the adversary capabilities, as detailed next.

System and Adversary Model Extensions Supporting TEE. We extend the adversary model of [Norrman et al., 2021] by allowing the adversary to use the long term key of any party via an interface, without directly accessing the key itself. This allows us to model the scenario where the adversary has gained access to a device, but the long-term key is protected by a trusted execution environment (TEE), and thus only accessible via the TEE interface. According to the terminology of [Cohn-Gordon et al., 2016], a protocol is considered to enjoy *weak post-compromise security* (Weak-PCS) if it achieves its security goals for a specific session even when the following hold:

- Before the session starts, the adversary has limited access to the long-term keys of the parties involved, through an interface that securely maintains the keys but allows principals to run cryptographic operations using them, and
- The adversary has full access to the keys of the parties involved after the end of the session, as well as access to the keys of all other parties.

As in the framework of [Xu et al., 2020], we add adversary capabilities corresponding to a server adversary, i.e., upon compromising a party, the adversary learns their ephemeral keys and may temporarily access their TEE, but does not learn their long-term key. This extension to the model of [Norrman et al., 2021] thus allows us to verify all the previous security properties under a "Weak-PCS model". Note that this attacker model is strictly more powerful than that of [Norrman et al., 2021], as it subsumes the previous attacker capabilities.

We split the functionality of EDHOC as follows. The TEE contains the long-term key and allows the non-TEE parts of the application to perform the operations using it via an interface. More precisely, parties using the SIG authentication method use a TEE

interface which accepts a message and returns the signature of that message using the party's private long-term key. A party U using the STAT authentication method uses a TEE with an interface accepting a point P on the curve and returning $dh(d_{s,U}, P)$.

This interface requires the least functionality from the TEE, reducing the TEE's complexity and (possibly) its cost. This functional split is suitable even when a constrained device has implemented the storage of only the long-term key in a special purpose circuit with minimal processing functionality. Since EDHOC focuses on constrained IoT devices, it seems appropriate to cater for this setting.

Extended Formalism and Security Properties. Formally, we model the TEE interface by adding two new rewrite rules:

```
rule forge_SIG:
   [!LTK_SIG($A, ~ltk), In(xx)] --[TEE($A)]-> [Out(sign(xx, ~ltk))]

rule exp_STAT:
   [!LTK_STAT($A, ~ltk), In('g'^~xx)] --[TEE($A)]-> [Out(('g'^~xx)^~ltk)]
```

These rules allow the adversary to obtain terms representing signatures on a value of their choice (`forge_SIG`), or to obtain terms representing a curve point of their choice raised to the power of the long-term key (`exp_STAT`).

Because it is a trivial attack when the adversary accesses these rules with values from a session in progress, we must disqualify those rule applications. We do so by creating an action fact `TEE($A)`, where `$A` is the identity corresponding to the private key used, and then augmenting the properties with a condition that no such action fact exists from the start of the protocol execution till its end. Care must be taken when specifying the start and the end: specifically, the start and end of the execution must be viewed with respect to the current role. For example, the injective agreement property for the initiator in the SIG-SIG method requires that the adversary does not have access to the TEE of the responder from the time that the first message is transmitted till the second message is received by the initiator (or equivalently, till the third message is transmitted by the initiator, since reception of a message and transmission of the next one by a party is one atomic operation). Thus, these timepoints represent the start and end of the protocol run from the perspective of the initiator.

The second message encrypts two values: the responder's identity `R` and the authentication information `authR`. We model XOR encryption by XOR-ing each term with their own key-stream term. However for some problematic methods[6] we XOR the entire tuple `<R, authR>` with a single key-stream term. This simplification is likely to miss an attack on implicit authentication which occurs due to the combination of a malleable XOR encryption and access to the TEE interface. However, given that no attacks were identified due to the use of XOR in our original modeling, nor in the current modeling for all other authentication methods, we believe that this is not a severe restriction.

Next we provide a quick overview of the Tamarin tool, and describe our modeling of EDHOC in it.

[6] See Table 2 for all verification results and computation times.

4.2 Tamarin

We extend the formal model of [Norrman et al., 2021], using the tool Tamarin [Meier et al., 2013], which is an interactive tool for the symbolic verification of security protocols. Protocols are modelled in Tamarin as multiset rewrite rules which encode a transition relation. The elements of these multisets, called facts, contribute to the global system state. For syntactic sugar, Tamarin also allows the use of let-bindings and tuples. For ease of presentation, we will present the model and properties in a slightly different syntax in this paper, but this syntax can be directly mapped to that of Tamarin.

Rewrite rules can be annotated with events, called actions in Tamarin. Communicated messages in the protocol are modelled as terms in an algebra, which specifies sets of names, variables, and allowable function symbols. Facts and actions are modelled as n-ary predicates in the term algebra, and actions can be parametrized using terms.

An annotated multiset-rewrite rule is represented as $l \,-[e]\!\!\rightarrow\, r$, where l and r are multisets, and e is a multiset of actions. A sequence of actions yields a protocol execution trace. Properties are defined as formulas in a fragment of temporal first order logic, and these formulas can be verified over execution traces. Event types are predicates over global states generated during protocol execution. Consider an event type E and a timestamp t as part of a trace. By $E^t(p_i)_{i\in\mathbb{N}}$, we represent an event of type E occurring at time t in a trace, parametrized by the sequence of values $(p_i)_{i\in\mathbb{N}}$ (corresponding to the action fact $E(p_i)_{i\in\mathbb{N}}@t$ in Tamarin). Thus, the time points form a quasi order, and we denote the fact that t_1 comes before t_2 in a protocol trace by $t_1 \lessdot t_2$, and that t_1 and t_2 stand for the same time point in a trace by $t_1 \doteq t_2$. Tamarin allows events to occur at the same time point, with one restriction: multiple events of the same type cannot occur simultaneously, so if $t_1 \doteq t_2$, then $E^{t_1} = E^{t_2}$.

Protocol verification in Tamarin happens under an equational theory E. For example, to represent the fact that E satisfies the reversal of symmetric encryption by using a decryption operation with the key, one can write $dec(enc(x,y),y) =_E x$. The equational theory E is fixed upfront to handle the functions supported by the term algebra, so we will omit the subscript for the rest of this paper.

Users can extend the default term algebra and equational theory in Tamarin with new function symbols and unification rules (if any) for these new symbols. For example, EDHOC requires authenticated encryption, which we model using the symbol `aeadEncrypt`. We augment Tamarin with the following rule for this operation, which represents the fact that if the adversary knows a key `k`, a message `m`, and authenticated data `ad`, and has access to an encryption algorithm `ai`, then they can obtain the message corresponding to the authenticated encryption of `m` with `k` [Norrman et al., 2021].

```
[!KU(k), !KU(m), !KU(ad), !KU(ai)] --[]-> [!KU(aeadEncrypt(k, m, ad, ai))]
```

Other than modeling authenticated encryption, we use Tamarin's built-in equational theories for signing, Diffie-Hellman, hashing and XOR.

Tamarin has also built-in rules for modeling a Dolev-Yao adversary and the evolution of their knowledge as the protocol executes. We extend the Dolev-Yao adversary model by adding other rules that increase the capabilities of the attacker. To denote that the adversary has access to a message p at time t, we use $\mathcal{K}^t(p)$. As an example, the following implication

$$\forall t, k, k' \,.\, \mathcal{K}^t(\langle k, k' \rangle) \;\rightarrow\; \mathcal{K}^t(k) \wedge \mathcal{K}^t(k'),$$

models the fact that if the adversary gets to know the pair of keys $\langle k, k' \rangle$ at a time point t, then the adversary knows each of those keys k and k' at time point t as well. For more details about how Tamarin manages adversary knowledge, see [Meier et al., 2013].

4.3 Model and Desired Properties

In this section, we describe our modeling of EDHOC and its desired security properties.

The party I executing the initiator role considers a run of the protocol begun as soon as it sends the first message m_1 with event type $\mathbf{I}_\mathsf{S}$, and considers the run ended once it has sent the third message m_3 with event type $\mathbf{I}_\mathsf{C}$. Similarly, the responder R considers a run started upon receiving m_1 with event type $\mathbf{R}_\mathsf{S}$, and finished upon receiving m_3 with type $\mathbf{R}_\mathsf{C}$.

In this work we consider the following properties: secrecy of the session key, injective agreement and implicit agreement on the session-key material for both initiator and responder, and secrecy and integrity of the application data sent as part of message 3 (ad_3). Agreement is considered on a set of parameters S which also contains the session-key material Z. We will first describe in detail all these properties, and then describe the contents of the set S. We formalize these properties as shown in Figure 4, which is a modified version of a figure in [Norrman et al., 2021].

Secrecy of the Session-Key Material. We show that the adversary cannot gain access to the session-key material Z, even under the weak post-compromise security model which allows the adversary limited access to the long-term keys of the involved parties via a secure interface before the session starts, and unrestricted access to the long-term keys of all parties as well as access to all other session keys after the session ends. This property is formalized as **PCS** in Figure 4.

In the $\mathbf{I}_\mathsf{C}$ event, *tid* represents a "thread identifier" which uniquely identifies the current session, I represents the identity of the initiator, and R represents the identity of the party who the initiator believes is playing the responder role, while Z stands for the established session-key material. $\mathbf{R}_\mathsf{C}$ has analogous parameters; in particular, the responder R believes I is the party playing the initiator role. Intuitively, the PCS property states that an adversary obtains Z only if one of the following conditions hold: either a party's long-term key is compromised before their run ends, or the TEE interface for the responder is used during the current session for the initiator, i.e., after the initiator starts and before it completes, or, similarly, the TEE interface for the initiator is used during the current session for the responder. Note that the properties in Figure 4 are unlike the actual Tamarin lemma in one minor way: Tamarin's logic does not allow disjunctions to appear on the left-hand side of an implication inside a universally-quantified formula. Therefore, in the Tamarin code, instead of using the disjunction $\mathbf{I}_\mathsf{C}^{t_2}(tid, I, R, Z) \vee \mathbf{R}_\mathsf{C}^{t_2}(tid, I, R, Z)$ to model the fact that either party may have completed their execution, we use a single action parametrized by the terms *tid*, I, R, and Z.

Authentication Properties. Following [Norrman et al., 2021], we prove two different kinds of authentication properties, namely *injective agreement* in the style of [Lowe, 1997], and implicit agreement. Injective agreement can be guaranteed to either party

$$\begin{aligned}
\textbf{PCS} \triangleq\ & \forall tid, I, R, Z, t_2, t_3 \,.\, \mathcal{K}^{t_3}(Z) \ \wedge\ (\mathbf{I}_{\mathsf{C}}^{t_2}(tid, I, R, Z) \vee \mathbf{R}_{\mathsf{C}}^{t_2}(tid, I, R, Z)) \rightarrow \\
& (\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(I) \wedge t_1 \lessdot t_2) \ \vee\ (\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(R) \wedge t_1 \lessdot t_2) \\
& \vee\ (\exists t_1 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_1}(R, Z)) \ \vee\ (\exists t_1 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_1}(I, Z)) \\
& \vee\ (\exists t_0, t_1, Z' \,.\, \mathbf{I}_{\mathsf{S}}^{t_0}(tid, I, R, Z') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(R) \wedge (t_0 \lessdot t_1 \wedge t_1 \lessdot t_2)) \\
& \vee\ (\exists t_0, t_1, Z' \,.\, \mathbf{R}_{\mathsf{S}}^{t_0}(tid, I, R, Z') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(I) \wedge (t_0 \lessdot t_1 \wedge t_1 \lessdot t_2))
\end{aligned}$$

$$\begin{aligned}
\textbf{InjAgree}_I \triangleq\ & \forall tid_I, I, R, Z, S, t_2 \,.\, \mathbf{I}_{\mathsf{C}}^{t_2}(tid_I, I, R, Z, S) \rightarrow \\
& (\exists tid_R, t_1 \,.\, \mathbf{R}_{\mathsf{S}}^{t_1}(tid_R, R, Z, S) \wedge t_1 \lessdot t_2 \wedge (\forall tid_I', I', R', t_1' \,.\, \mathbf{I}_{\mathsf{C}}^{t_1'}(tid_I', I', R', Z, S) \rightarrow t_1' \doteq t_2)) \\
& \vee\ (\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(R) \wedge t_1 \lessdot t_2) \\
& \vee\ (\exists t_0, t_1, Z' \,.\, \mathbf{I}_{\mathsf{S}}^{t_0}(tid_I, I, R, Z') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(R) \wedge (t_0 \lessdot t_1 \wedge t_1 \lessdot t_2))
\end{aligned}$$

$$\begin{aligned}
\textbf{InjAgree}_R \triangleq\ & \forall tid_R, I, R, Z, S, t_2 \,.\, \mathbf{R}_{\mathsf{C}}^{t_2}(tid_R, I, R, Z, S) \rightarrow \\
& (\exists tid_I, t_1 \,.\, \mathbf{I}_{\mathsf{S}}^{t_1}(tid_I, I, R, Z, S) \wedge t_1 \lessdot t_2 \wedge (\forall tid_R', I' R' t_1' \,.\, \mathbf{R}_{\mathsf{C}}^{t_1'}(tid_R', I', R', Z, S) \rightarrow t_1' \doteq t_2)) \\
& \vee\ (\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(I) \wedge t_1 \lessdot t_2) \\
& \vee\ (\exists t_0, t_1, Z', S' \,.\, \mathbf{R}_{\mathsf{S}}^{t_0}(tid_R, R, Z', S') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(I) \wedge (t_0 \lessdot t_1 \wedge t_1 \lessdot t_2))
\end{aligned}$$

$$\begin{aligned}
\textbf{ImpAgree}_I \triangleq\ & \forall tid_I, I, R, Z, S, t_1 \,.\, \mathbf{I}_{\mathsf{C}}^{t_1}(tid_I, I, R, Z, S) \rightarrow \\
& (\forall tid_R, I', R', S', t_2 \,.\, \mathbf{R}_{\mathsf{C}}^{t_2}(tid_R, I', R', Z, S') \rightarrow (I = I' \wedge R = R' \wedge S = S') \\
& \wedge (\forall tid_I', I', R', S', t_1' \,.\, \mathbf{I}_{\mathsf{C}}^{t_1'}(tid_I', I', R', Z, S') \rightarrow t_1' \doteq t_1)) \\
& \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_0}(R) \wedge t_0 \lessdot t_1) \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(R, Z)) \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(I, Z)) \\
& \vee\ (\exists t_0, t_1, Z' \,.\, \mathbf{I}_{\mathsf{S}}^{t_0}(tid_I, I, R, Z') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(R) \wedge (t_0 \lessdot t_1 \wedge t_1 \lessdot t_2))
\end{aligned}$$

$$\begin{aligned}
\textbf{Sec}ad_3 \triangleq\ & \forall ad_3, I, R, Z, tid_I, t_3, t_2 \,.\, \mathcal{K}^{t_3}(ad_3) \wedge \mathbf{I}_{\mathsf{SEND}}^{t_2}(I, R, Z, ad_3, tid_I) \rightarrow \\
& ((\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(I) \wedge t_1 \lessdot t_2) \vee (\exists t_1 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_1}(R) \wedge t_1 \lessdot t_2) \\
& \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(R, Z)) \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(I, Z)) \\
& \vee (\exists t_0, t_1, Z' \,.\, \mathbf{I}_{\mathsf{S}}^{t_0}(tid_I, I, R, Z') \wedge \mathbf{A}_{\mathsf{TEE}}^{t_1}(R) \wedge t_0 \lessdot t_1) \\
& \vee (\exists t_0, t_1, t_4, tid_R, Z', S' \,.\, \mathbf{A}_{\mathsf{TEE}}^{t_1}(I) \wedge \mathbf{R}_{\mathsf{Recv}}^{t_4}(I, R, Z, ad_3, tid_R) \wedge \mathbf{R}_{\mathsf{S}}^{t_0}(tid_R, R, Z', S') \wedge t_0 \lessdot t_1))
\end{aligned}$$

$$\begin{aligned}
\textbf{Integ}ad_3 \triangleq\ & \forall ad_3, I, R, Z, tid_R, t_3 \,.\, \mathbf{R}_{\mathsf{Recv}}^{t_3}(I, R, Z, ad_3, tid_R) \rightarrow \\
& ((\exists tid_I, t_2 \,.\, \mathbf{I}_{\mathsf{SEND}}^{t_2}(I, R, Z, ad_3, tid_I) \wedge t_2 \lessdot t_3) \\
& \vee (\exists t_2 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_2}(I) \wedge t_2 \lessdot t_3) \vee (\exists t_2 \,.\, \mathbf{A}_{\mathsf{LTK}}^{t_2}(R) \wedge t_2 \lessdot t_3) \\
& \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(R, Z)) \vee (\exists t_0 \,.\, \mathbf{A}_{\mathsf{Eph}}^{t_0}(I, Z)) \\
& \vee (\exists t_0, t_1, t_2, tid_I, Z' \,.\, \mathbf{A}_{\mathsf{TEE}}^{t_1}(R) \wedge \mathbf{I}_{\mathsf{SEND}}^{t_2}(I, R, Z, ad_3, tid_I) \wedge \mathbf{I}_{\mathsf{S}}^{t_0}(tid_I, I, R, Z') \wedge t_0 \lessdot t_1) \\
& \vee (\exists t_0, t_1, Z', S' \,.\, \mathbf{A}_{\mathsf{TEE}}^{t_1}(I) \wedge \mathbf{R}_{\mathsf{S}}^{t_0}(tid_R, R, Z', S') \wedge t_0 \lessdot t_1))
\end{aligned}$$

Fig. 4. Formalization of security properties

running the protocol. For the initiator I, it guarantees to I that whenever I believes that they have completed a run with R as responder, then the party R has indeed executed the protocol in the role of a responder, and that this run of I uniquely corresponds to one of R where the set of parameters is S and includes, in particular, the session-key material Z. It can be defined for R in a similar manner.

We formalize injective agreement for the initiator role as $\mathbf{InjAgree}_I$ and for the responder role as $\mathbf{InjAgree}_R$ in Figure 4. For the initiator I, this property represents the fact that either injective agreement (as described above) holds, or the long-term key of the party R assumed to be playing the responder role has been compromised before I's role finished. The property should hold even if the adversary has access to the TEE of party R before and after the protocol execution. An analogous definition holds for the responder R.

As part of our analysis, we found that all the EDHOC methods satisfy weak post-compromise security. However, this is not the case for the injective agreement property as stated above. Thus, we show a different property, a form of implicit agreement on the same set of parameters, which is guaranteed for all methods. This modification is inspired by the definitions of implicit authentication in the computational model [Delpech de Saint Guilhem et al., 2020]. While that paper focuses on authenticating just a key and related identities, our definition encompasses a general set of parameters, as in the notion of injective agreement proposed by Lowe [Lowe, 1997].

The "implicit" in the name of the property stands for the fact that a party A assumes that any party B who has access to the session-key material Z must, in fact, be the intended party, and that if B is honest, B will agree on a set S of parameters which includes Z. Implicit agreement for both roles guarantees to A that A is or has been involved in exactly one protocol execution with B, and that B agrees or will agree with A on S. This property diverges from injective agreement in that upon sending the last message, A concludes that if this message reaches B, then A and B agree on each other's identities and roles, as well as the set S.

Note that for implicit agreement to hold for I, the ephemeral keys must not be revealed since the property relies on the fact that the intended responder is the only one who knows the session-key material. If the adversary has access to the ephemeral keys, they can use them along with the public keys of I and R to compute the session-key material. However, either party's long-term key can be revealed after the other party has finished their execution, since this still leaves the adversary unable to compute P_e.

Since Tamarin runs out of memory to verify this property as is, we split it into two lemmas – $\mathbf{ImpAgree}_I$ for I for one $\mathbf{ImpAgree}_R$ for R. Figure 4 contains the definition for $\mathbf{ImpAgree}_I$. $\mathbf{ImpAgree}_R$ is formalized similarly, so we omit it.

Set S of Agreed Parameters. We now describe the set S of parameters upon which the two parties obtain guarantees via the above properties. The initiator I gets injective and implicit agreement guarantees on the following partial set S_P of parameters [Norrman et al., 2021]:

- the roles played by itself and its peer,
- responder identity,
- session-key material (which varies depending on the EDHOC method),

- context identifiers C_I and C_R, and
- cipher suites S_I.

Due to the initiator being guaranteed identity protection under EDHOC, I cannot get explicit agreement with R on the initiator's identity. Similarly, when using the STAT authentication method, I does not get any such guarantees about P_I. However, I does get implicit agreement with R about I's identity and the full set S_F of agreed parameters. In contrast, since R's run finishes after that of I, R can get explicit injective agreement assurances on the full set S_F of agreed parameters. The full set of agreed parameters S_F is $S_P \cup \{I, P_I\}$ when P_I is part of the session-key material, and $S_P \cup \{I\}$ otherwise.

In addition to these properties, a couple of properties can be inferred without being explicitly modelled and verified. One such property is Key-Compromise Impersonation (KCI) [Blake-Wilson et al., 1997]. A KCI attack occurs when an adversary who has access to A's long-term private key to make A believe that they completed an execution with a peer B, while B did not participate in said execution at all. This is in particular relevant when STAT authentication methods are used. Our above notions of agreement ensure that both parties agree on each other's identity, role, and session-key material. Therefore, all EDHOC methods that satisfy these agreement properties also avoid KCI attacks.

Another kind of attack is Unknown Key Share attacks (UKS) [Blake-Wilson et al., 1997]. As part of a UKS attack, a party A can be made to believe that it finished an execution with a party B, but where the session-key material is actually shared between A and C instead. Again, due to the agreement on identities and session-key material, any method that satisfies the above agreement properties also resists UKS attacks. Overall, the agreement properties capture entity authentication, and satisfy any properties based on that notion. However, see Section 6.1 for a discussion on the interaction between EDHOC and the application leading to similar issues.

4.4 Encoding EDHOC in Tamarin

In this section, we describe how we model the SIG-SIG, SIG-STAT, STAT-SIG, and STAT-STAT methods of EDHOC in Tamarin. As in [Norrman et al., 2021], we construct the Tamarin model by utilizing the fact that all methods of EDHOC share a common underlying structure (as shown in Figure 1). We do so by using a single specification file in the M4 macro language, which generates all the methods. As in [Norrman et al., 2021], we only present the STAT-SIG method which illustrates the use of two different authentication methods. The full Tamarin code for all models can be found at [Norrman et al., 2022].

As mentioned in Section 4.2, we extend the default equational theory of Tamarin to handle various operations used in EDHOC. Of the built-in theories, we use the ones for exclusive-or (XOR), Diffie-Hellman operations, signatures (`sign` and `verify` operations), and hashing [Dreier et al., 2018,Schmidt et al., 2012].

In addition to these default operations, EDHOC is built over the following functions: HKDF-expand, HKDF-extract, and AEAD. We represent HKDF-expand by `expa` and HKDF-extract by `extr`. AEAD operations need us to add extra equations to the underlying theory. A term encrypted using the AEAD algorithm is represented by

`aeadEncrypt(m, k, ad, ai)`, where `m` is the underlying message, `k` is the encrypting key, `ad` is the additional data, and `ai` is the identifier for the encryption algorithm. Decryption of such a term is defined via two equations that we add to Tamarin's theory. The following equation requires the decrypting party to know the additional data `ad` to decrypt this encrypted term with verification of its integrity.

```
aeadDecrypt(aeadEncrypt(m, k, ad, ai), k, ad, ai) = m.
```

Only the above equation is used by honest parties, but the adversary should also be able to decrypt without having to go through the additional step of identity verification. To this end, we also add the following equation, where the adversary does not need access to the additional data `ad` in order to decrypt.

```
decrypt(aeadEncrypt(m, k, ad, ai), k, ai) = m.
```

Having described how we adapt the equational theory to model EDHOC, we now move on to the modeling of the adversary model and the environment in which the protocol is executed. We extend the built-in Dolev-Yao adversary rules which are part of Tamarin. We use the following rules to capture the link between a party's identity and their long-term key pairs, in the SIG- and in the STAT-based methods respectively.

```
1 rule registerLTK_SIG:
2    [Fr(~ltk)] --[UniqLTK($A, ~ltk)]->
3        [!LTK_SIG($A, ~ltk),
4         !PK_SIG($A, pk(~ltk)),
5         Out(<$A, pk(~ltk)>)]
```

```
1 rule registerLTK_STAT:
2    [Fr(~ltk)] --[UniqLTK($A, ~ltk)]->
3        [!LTK_STAT($A, ~ltk),
4         !PK_STAT($A, 'g'^~ltk),
5         Out(<$A, 'g'^~ltk>)]
```

Using the fact `Fr(~ltk)`, Tamarin creates a new term `ltk` and uses it to represent a secret long-term key. Via the `Out(<$A, pk(~ltk)>)` fact, Tamarin puts out onto the communication channel the identity of the party to whom this long-term key belongs, along with their public key. Since the adversary has access to the communication channel, they can pick up all of this information. The event `UniqLTK` parametrized by a party's identity and their long-term key models the unique correspondence between those two values. As a result, this rules out a party owning multiple long-term keys – in particular, it keeps the adversary from registering long-term keys in some honest party's name. This aligns well with an external mechanism such as a certificate authority ensuring that long-term keys are uniquely assigned to the corresponding identities, which is ensured by the specification.

To model the reveal of long-term keys and ephemeral keys to an adversary, we use standard reveal rules and events of type $\mathbf{A}_{\mathsf{LTK}}$ and $\mathbf{A}_{\mathsf{Eph}}$, respectively. It is also important to keep track of the time points at which these events occur. Long-term keys can be revealed on registration, even before the protocol begins. Ephemeral keys in our model can only be revealed when a party completes their role, i.e., simultaneously with events of type $\mathbf{I}_{\mathsf{C}}$ and $\mathbf{R}_{\mathsf{C}}$. Having set out the capabilities of the adversary, we now model the execution of the honest agents' roles.

For each protocol method, we use two rules apiece for the initiator and responder – `I1`, `R2`, `I3`, `R4`. Each of these stand for one step of the protocol as executed by either party. To disambiguate, we will attach the method to the rule name. These four rules directly map to the event types $\mathbf{I}_{\mathsf{S}}$, $\mathbf{R}_{\mathsf{S}}$, $\mathbf{I}_{\mathsf{C}}$, and $\mathbf{R}_{\mathsf{C}}$, respectively. We show the `R2_STAT_SIG` rule below to illustrate the various aspects of the Tamarin modeling we are describing here.

In order to keep track of the initiator's state, we use facts prefixed with `StI`, which carry information between the `I1` and `I3` rules. Similarly, for the responder's state, we have `StR` to carry state data between `R2` and `R4`. In order to link two rules to a state fact, we use `tid`, which is unique to the current session. The use of these state facts can be seen in line 28 in the `R2_STAT_SIG` rule.

Note that we do not model any error message that R might send in response to message m_1 rejecting I's choice of ciphersuite and/or method.

As in [Norrman et al., 2021], we model the XOR encryption of `CIPHERTEXT_2` with the key `K_2e` by allowing each part of the encrypted term to be separately attacked. This means that we first expand `K_2e` to the same number of key terms as subterms in the plaintext tuple. This is done by applying HKDF-expand to unique inputs per subterm. After this, we XOR each subterm with its own key term. This is more faithful to the specification than XOR-ing `K_2e` on its own with the plaintext tuple. This can be seen in lines 18–21 in the code for `R2_STAT_SIG`.

As we extended the model with TEEs and augmented the adversary's capabilities with access to them, Tamarin failed to complete in a reasonable time for some combination of authentication methods and security properties (see Section 7 for a detailed discussion). To circumvent the problem, we simplified the XOR encryption to XOR-ing a single term on the entire tuple for these cases.

```
1 rule R2_STAT_SIG:
2 let
3    agreed = <CS0, CI, ~CR>
4    gx = 'g'^xx
5    data_2 = <'g'^~yy, CI, ~CR>
6    m1 = <'STAT', 'SIG', CS0, CI, gx>
7    TH_2 = h(<$H0, m1, data_2>)
8    prk_2e = extr('e', gx^~yy)
9    prk_3e2m = prk_2e
10   K_2m = expa(<$cAEAD0, TH_2, 'K_2m'>, prk_3e2m)
12   protected2 = $V // ID_CRED_V
13   CRED_V = pkV
14   extAad2 = <TH_2, CRED_V>
15   assocData2 = <protected2, extAad2>
16   MAC_2 = aead('e', K_2m, assocData2, $cAEAD0)
17   authV = sign(<assocData2, MAC_2>, ~ltk)
18   plainText2 = <$V, authV>
19   K_2e = expa(<$cAEAD0, TH_2, 'K_2e'>, prk_2e)
20   K_2e_1 = expa(<$cAEAD0, TH_2, 'K_2e', '1'>, prk_2e)
21   K_2e_2 = expa(<$cAEAD0, TH_2, 'K_2e', '2'>, prk_2e)
22   CIPHERTEXT_2 = <$V XOR K_2e_1, authV XOR K_2e_2>
23   m2 = <data_2, CIPHERTEXT_2>
24   exp_sk = <gx^~yy>
25   in
26   [!LTK_SIG($V, ~ltk), !PK_SIG($V, pkV), In(m1), Fr(~CR), Fr(~yy), Fr(~tid)]
27   --[ExpRunningR(~tid, $V, exp_sk, agreed), R2(~tid, $V, m1, m2)]->
28   [StR2_STAT_SIG($V, ~ltk, ~yy, prk_3e2m, TH_2, CIPHERTEXT_2, gx^~yy, ~tid, m1, m2, agreed),
29    Out(m2)]
```

As mentioned earlier, we use actions to represent parametrized events. For example, in line 27 above, the action `ExpRunningR(~tid, $V, exp_sk, agreed)` represents an event of type $\mathbf{R}_S$ parametrized by the session id, the responder's identity, and the session-key material `exp_sk`. The `exp` in the name of the variable for session-key material represents the fact that the agreement property satisfied by this key is explicit, i.e., it includes P_I, as in Section 4.3. We use `imp_sk` for the corresponding session-key

material which does not include P_I. For the SIG-SIG and SIG-STAT methods, therefore, the two values are the same.

The properties we listed in Section 4.3 translate directly into Tamarin's logic. We show the Tamarin lemma which encodes the **PCS** property. Other properties are formalized similarly.

```
1 lemma secrecyPCS:
2    all-traces
3    "All u v sk tid #t3 #t2.
4       (K(sk)@t3 & CompletedRun(u, v, sk, tid)@t2) ==>
5       ( (Ex #t1. LTKRev(u)@t1 & #t1 < #t2)
6       | (Ex #t1. LTKRev(v)@t1 & #t1 < #t2)
7       | (Ex #t1. EphKeyRev(sk)@t1)
8       | (Ex m1 #t0 #k. I1(tid, u, v, m1)@t0 & TEE(v)@k & (t0 < k | k < t2))
9       | (Ex m1 m2 #t0 #k. R2(tid, v, m1, m2)@t0 & TEE(u)@k & (t0 < k | k < t2))
10    )"
```

In this formalization, we use the action `CompletedRun(u, v, sk)` (in line 4) to represent the disjunction of the events $\mathbf{I}_{\mathsf{C}}^{t_2}$ and $\mathbf{R}_{\mathsf{C}}^{t_2}$. As expected, this action is emitted by both `I3` and `R4`. Similarly, the action `EphKeyRev(sk)` in line 7 stands for the reveal of the ephemeral key for I or R or both. Lines 8 and 9 captures that the parties TEEs must be inaccessible to the adversary between the start and end of the execution as seen by each party respectively. The entire code can be found at [Norrman et al., 2022].

5 ERROR HANDLING

Section 6 of the current EDHOC specification [Selander et al., 2022] states that error messages can be sent at any time and by any party. There are three types of error messages: type 0 is used to represent success, type 1 represents a generic error message used for debugging purposes, and type 2 represents the failure to negotiate a common ciphersuite. Type 2 messages carry a list of supported ciphersuites by the responder. The contents and semantics of type 1 messages are dependent on the implementation. Consequently, type 1 error messages provide a generic message passing mechanism, albeit without a predefined semantics. We now discuss some issues with this specification of error messages. In this section, we use "specification" to mean the latest specification, as in [Selander et al., 2022].

5.1 Lack of Connection Identifiers

Error messages do not include the optional connection identifiers C_I or C_R. This is potentially problematic for connection-less transport layers, where these identifiers may be used to correlate messages.

5.2 Proper Reception Handling

EDHOC provides an error message mechanism, but little to no guidance on how it should be used safely. For example, assume that an application using EDHOC logs error messages on a finite log with rotation. If the log is used for anomaly detection or detection of sensitive events, then an adversary can simply inject error messages and fill the log until the sensitive event is overwritten. In such situations, implementers should do proper log-separation and log management.

5.3 Unspecified Semantics for Debugging Messages

Section 6.2 gives an example of sending a type 1 error message with the information string “Method not supported". Note that EDHOC does not include a mechanism to support the negotiating of the connection method. The initiator selects the method as part of the first message, and if the responder replies with a type 1 error message along with the information string “Method not supported", the standard does not tell an implementation how to act upon this information string for negotiation purposes. An adversary can fake an error message, and there is no algorithm that is specified to ensure that a common available authentication mechanism is always selected. In particular, note that Section 5.1 in the specification states that if processing fails for some reason, then “typically" an error message is sent, processing is terminated, and protocol state is discarded. But negotiation needs us to remember state.

With respect to ciphersuite negotiation, Section 5.2.2 of the specification also states “The Initiator MUST NOT change the supported cipher suites and the order of preference in SUITES_I based on previous error messages.” However, S_I is sent in the clear, and therefore can be modified by the adversary. It is important to consider what happens if the order of preference in S_I is changed either by the adversary, or by the initiator themselves for some other reason. It might be helpful to include a flag in m_1 to indicate to the responder that a (modified) ciphersuite ordering depends on that from a previous execution.

6 IMPROVEMENTS TO THE SPECIFICATION

The EDHOC specification has many points where it can be improved. We discovered a few of these issues in [Norrman et al., 2021], but in this work, we also discuss a couple of new issues. We communicated the points discussed in [Norrman et al., 2021] to the authors of the specification. Here, we also list the protocol developers' response (if any) to being told of these ambiguities in the specification.

6.1 Unclear Intended Use

There are several security goals listed in the specification. However, these goals are informal and imprecisely specified, and the lack of intended usage makes them difficult to interpret. Since we have little indication about use case restrictions, it is also hard to evaluate whether the security goals are the most relevant ones.

In order to identify security goals that might be relevant we made up some user stories corresponding to typical use cases, which helped us identify some subtle points of concern in EDHOC. We communicated these points to the authors of EDHOC.

Non-Repudiation. Access control mechanisms can vary depending on what the underlying application is. A nuclear power-plant might need to keep track of who enters and leaves, but a coffee machine that logs every user and their coffee preference might lead to a privacy concern. This simple thought experiment allowed us to identify that the EDHOC specification did not even consider non-repudiation. When we pointed this out

to the authors of the specification, they recognized this concern, and added a discussion about how the various EDHOC methods satisfy (or not) non-repudiation.

Unintended Peer Authentication. In Section 3.2 in the specification, the authors state that parties executing EDHOC must be configured such that they only run EDHOC with a restricted (according to some reasonable policy) set of peers. However, the responsibilities of verifying which identity was authenticated is not clearly split between the application and EDHOC and therefore an attack similar in effect to UKS is possible.

Suppose a person configures the restriction policy such that every device in their home is allowed to set up an EDHOC session with any other device in the home. However, the adversary has managed to gain control of one of these devices (say A). Device A is part of the allowed group of devices, so B will accept an EDHOC run with A, even though A is now compromised. If the application in B tasks the EDHOC implementation to establish a security context with C, B will send an initial message to C. However, if A responds to the initial message before C can, the specification does not require B to verify that ID_R matches what is expected for C. A valid EDHOC implementation may then look up the appropriate credentials for A, complete the execution and deliver the security context to the application. The application gets no indication of that the security context is shared with A rather that C as intended.

A straight forward check that the received ID_R matches the expected identity thwarts the problem. We included this check in our model. We communicated this to the authors of EDHOC, who in the latest version of the specification, added that EDHOC makes ID_R available to the application, and added an appendix D listing credential validations that the application must take care of.

Denial of Service. Denial of Service (DoS) attacks for IoT protocols are well-studied, but DoS aspects particular to EDHOC is not. The specification does not include any specific measures countering such attacks, but assumes the application takes care of it.

Section 8.7 of the latest version of the EDHOC specification [Selander et al., 2022] gives two recommendations for countering DoS attacks. First, EDHOC relies on lower layers to mitigate DoS, and an example of DoS countermeasures is given: checking the return address upon receipt of the first message. Second, a recommendation is given that applications may try to determine (in some way) that a seemingly valid message is in fact probably a forgery and should be ignored or cause the protocol to abort.

No recommendation is given on how to minimize DoS effects stemming from the use of error messages, which can be used in a relatively arbitrary way in EDHOC (see Section 5). We now elaborate on some considerations that the application developers need to take into account.

As in the IKEv2 DDoS analysis by [Nir and Smyslov, 2016], the responder in EDHOC is more vulnerable than the initiator. The key similarity is that the initiator enjoys identity protection in the first message. None of the elements in the first EDHOC message can be proved by the responder to be invalid, as long as the format is correct and the chosen elliptic curve points lie on the chosen curve. Additionally, execution may require the responder to obtain and verify certificate chains. For example, the content of the ad_1 information element may need the responder to take further action. Since

EDHOC is designed for constrained devices, this asymmetry needs to be considered even more carefully than for IKEv2 [Nir and Smyslov, 2016].

The asymmetry between the adversary's effort towards the first message and that of the responder may hence be significant and the possibility for applications to select which entity acts as initiator and which acts as responder may important. EDHOC is profiled to work in conjunction with CoAP and OSCORE [Palombini et al., 2022]. The profiling locks the EDHOC initiator role to the CoAP client role and hence reduces the possibilities to change the EDHOC roles according to DoS vulnerability preferences.

Reflection attacks are another risk. These are attacks where an adversary spoofs the source address of a victim in a message and thereby tricks the responder into sending a message to the victim. EDHOC suggests using a return path reachability method similar to IKEv2. Note that even such reachability mechanisms can easily be used to make a responder send reachability requests to arbitrary addressable targets. However, such mechanisms might require that unsolicited reachability requests be discarded, limiting the effects of anonymous DoS attacks. In situations where the responder does not need to perform additional communications to verify certificates etc, a reachability mechanism may be more expensive in terms of time and storage compared to continuing with the protocol while keeping a half-open state for a period of time. However, if round-trip times are large, and half-open states hence are kept by the receiver for a long time, a reachability mechanism can trade communication for storage.

EDHOC responders querying external servers for certificate revocation checks or lookups may cause systems-level issues if many distributed EDHOC responders frequently queries such servers when DoSed by clients.

An adversary may also send forged error messages (see Section 5), and specifically reject proposed ciphersuites. The EDHOC specification recommends the initiator to not try that ciphersuite with the responder again in a new session. If the adversary rejects all options for an initiator this way, it can prevent communication between the initiator and that responder until the initiator's cache is cleared. Potentially, an adversary could lock a party of a group out of communication altogether this way.

6.2 Unclear Security Model

In addition to unclear contexts in which EDHOC can be used, the specification also does not provide enough detail about adversary capabilities. The four different cryptographic cores included in EDHOC are based on academic protocols which are designed to work with highly specific but potentially different adversary models. Since SIGMA cannot protect against compromised ephemeral keys, the authors of EDHOC felt that considering the compromise of ephemeral keys separately from that of long-term keys was not required [EDHOC authors, 2020]. The reason was presumably based on the fact that the SIG-SIG method was modeled closely on the SIGMA-I variant of SIGMA, and that it would be preferable to obtain a homogeneous security level among the EDHOC methods [Norrman et al., 2021]. However, this only holds true for situations where one is only interested in the session key confidentiality for an ongoing session. Secure modules, in addition to being able to store long-term keys, are useful in other ways too. They also provide weak Post-Compromise Security (PCS) guarantees as discussed

above. Upon discussion with the authors, they included recommendations on storing long-term keys, and how to perform operations on these keys inside a secure module.

6.3 Session Key Material

As we have seen, EDHOC establishes some session-key material, which can be fed into the EDHOC-Exporter to derive session keys. This key material is directly influenced by P_e and a party's secret static long-term key, if they use STAT. As mentioned earlier, mutual explicit injective agreement cannot be obtained for P_I. This might prove to be a problem, and we proposed three potential alternatives for addressing this to the EDHOC authors.

- Include ID_I or a hash thereof in the first and second messages: This increases message size and also leaks the initiator's identity.
- Derive session-key material without using P_I: This deviates from the structure of protocols like OPTLS from which the STAT-based methods draw heavy inspiration. In those protocols, including P_I while computing session-key material is crucial to obtaining resistance against the compromise of the initiator's ephemeral key.
- Include a fourth message from the responder to the initiator which contains a MAC based on information obtained using P_I:

While the last option increases the number of messages in the protocol, the EDHOC authors decided to go with this option for situations where it is necessary to provide explicit injective agreement on P_I.

7 CONCLUSIONS AND FUTURE WORK

As part of this paper, we modeled all four authentication methods of the EDHOC protocol using the Tamarin verification tool. We used a server with 2 Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz, which had a total of 32 cores / 64 threads, and of the total 384GiB, the server allotted 256GiB of memory for the Tamarin job. We report both the time taken to complete the task ("real time") and the CPU time spent on the task ("user time"). Time measurements are rounded to the nearest second. We formulated and verified the following properties in a precise adversary model with controlled access to TEEs and a precise modeling of XOR.

- Injective agreement for both I and R
- Implicit agreement for both I and R
- Perfect Forward Secrecy (PFS) for the session-key material
- Weak post-compromise security for the session-key material
- Secrecy and integrity of ad_3

We consider two separate sets of parameters, a partial set (S_P) and the full set (S_F). The set S_P consists of the following pieces of information: the identity of the responder, roles, context identifiers C_I and C_R, cipher suites S_I, and session-key material excluding P_I. For injective agreement, when the initiator uses a STAT method, they are ensured agreement only on this above set of parameters. The set S_F is the set S_P along with

Lemma \ Method	SIG-SIG	SIG-STAT	STAT-SIG	STAT-STAT
Injective agreement / I	S_F	S_F	S_P‡	S_P‡
Injective agreement / R	S_F	S_F	S_F	S_F
Time (real/user)	3m10.782s / 76m23.828s	52m57.153s / 1909m1.455s†	5m32.432s / 139m36.018s	30m54.891s / 848m36.833s
Implicit agreement / I	S_F	S_F	S_F	S_F
Implicit agreement / R	S_F	S_F	S_F	S_F
Time (real/user)	3m34.222s / 89m41.770s	6m46.187s / 180m6.615s	10m34.228s / 253m22.845s	1m32.187s / 67m2.723s †
Session key secrecy	✓	✓	✓	✓
Time (real/user)	1m23.317s / 52m16.347s	3m54.009s / 155m54.834s	12m33.884s / 457m32.698s	20m40.391s / 915m15.590s
Secrecy of AD3	✓	✓	✓	✓
Integrity of AD3	✓	✓	✓	✓
Time (real/user)	2m39.727s / 101m2.371s	5m51.728s / 223m59.728s	5m51.728s / 223m59.728s	46m0.643s / 1786m3.825s

Table 2. Verified properties. S_P contains roles, responder identity, session-key material (excluding P_I), C_I, C_R, and S_I. S_F is S_P, the initiator identity, and P_I.
†: For SIG-STAT and STAT-STAT implicit authentication using the full XOR model the verification had to be abandoned at 14246m43s and 2057m47s of computation time, respectively. The time reported is for verification under the simplified XOR modeling of message 2.
‡: When the initiator is using STAT mode we cannot get explicit agreement on the intiator's own key material, therefore in these two cases we check injective agreement for all key material except P_I.

the initiator's identity and P_I. The responder is guaranteed injective agreement on the parameters S_F, irrespective of what method they run, while the initiator is guaranteed injective agreement on S_F as long as they run a SIG method. Implicit agreement can be reached for both parties on the set S_F. In addition, mutual entity authentication, UKS- and KCI-resistance can be inferred from the verified properties. We present the results in Table 2.

Mutual entity authentication, UKS- and KCI-resistance can be inferred from the verified properties.

Further, we identified a situation where initiators may establish an OSCORE security context with a different party than the application using EDHOC intended, and proposed a simple mitigation. We discussed how the IETF may extract and better define security properties to enable easier verification. We also discussed some aspects of the protocol with respect to error handling and denial of service attacks.

We verified each method in isolation, and leave as future work to verify whether the methods are secure under composition.

7.1 A Note About the Version of EDHOC

In this work, we have analyzed the EDHOC specification as of July 2020 [Selander et al., 2020]. While our formal analysis applies to v5 of the specification (February 2021), there are newer versions. The latest version at the time of writing is v15 (July 2022) [Selander et al., 2022], which, among other things, differ from v5 in terms of some inputs to the key derivations. Modeling the new key derivation function would have needed sweeping changes across our entire formal model, and due to paucity of time, we could not modify the Tamarin model to be up to date with the current version. However, we do refer to various aspects (error handling, denial of service etc) of this

latest version, in Sections 5 and 6. Modeling the latest version of the protocol is also left as future work.

ACKNOWLEDGEMENTS

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We are grateful to Göran Selander, John Mattsson and Francesca Palombini for clarifications regarding the specification.

Bibliography

Blake-Wilson, S., Johnson, D., and Menezes, A. (1997). Key agreement protocols and their security analysis. In *Proc. of IMA Cryptography and Coding*, pages 30–45.

Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of IEEE CSFW-14*, pages 82–96.

Bruni, A., Jørgensen, T. S., Petersen, T. G., and Schürmann, C. (2018). Formal verification of ephemeral Diffie-Hellman over COSE (EDHOC). In *Proc. of SSR*, pages 21–36.

Cohn-Gordon, K., Cremers, C., and Garratt, L. (2016). On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 164–178. IEEE Computer Society.

Delpech de Saint Guilhem, C., Fischlin, M., and Warinschi, B. (2020). Authentication in key-exchange: Definitions, relations and composition. In *Proc. of IEEE CSF*, pages 288–303.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

Dreier, J., Hirschi, L., Radomirovic, S., and Sasse, R. (2018). Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In *Proc. of IEEE CSF*, pages 359–373.

EDHOC authors (2020). Personal communication.

Hristozov, S., Huber, M., Xu, L., Fietz, J., Liess, M., and Sigl, G. (2021). The cost of OSCORE and EDHOC for constrained devices. In Joshi, A., Carminati, B., and Verma, R. M., editors, *CODASPY '21: Eleventh ACM Conference on Data and Application Security and Privacy, Virtual Event, USA, April 26-28, 2021*, pages 245–250. ACM.

Krawczyk, H. (2003). SIGMA: The 'SIGn-and-MAc'approach to authenticated Diffie-Hellman and its use in IKE protocols. In *Proc. of CRYPTO*, pages 400–425.

Krawczyk, H. (2005). HMQV: A high-performance secure diffie-hellman protocol. In Shoup, V., editor, *Proc. of CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566.

Krawczyk, H. and Eronen, P. (2010). HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869.

Krawczyk, H. and Wee, H. (2016). The OPTLS protocol and TLS 1.3. In *Proc. of IEEE EuroS&P 2016*, pages 81–96.

Lowe, G. (1997). A hierarchy of authentication specification. In *Proc. of IEEE CSFW-10*, pages 31–44.

Meier, S., Schmidt, B., Cremers, C., and Basin, D. A. (2013). The TAMARIN prover for the symbolic analysis of security protocols. In *Proc. of CAV*, pages 696–701.

Nir, Y. and Smyslov, V. (2016). Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks. RFC 8019.

Norrman, K., Sundararajan, V., and Bruni, A. (2021). Formal analysis of EDHOC key establishment for constrained iot devices. In di Vimercati, S. D. C. and Samarati, P., editors, *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*, pages 210–221. SCITEPRESS.

Norrman, K., Sundararajan, V., and Bruni, A. (2022). Code repository. https://www.dropbox.com/sh/y3dk6t421040mq9/AADct4SSGOhOlbx6mFszvq-da?dl=0.

Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and Selander, G. (2022). Profiling EDHOC for CoAP and OSCORE. Internet-Draft draft-ietf-core-oscore-edhoc-03, Internet Engineering Task Force. Work in Progress.

Schmidt, B., Meier, S., Cremers, C. J. F., and Basin, D. A. (2012). Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Proc. of IEEE CSF*, pages 78–94.

Selander, G., Mattsson, J. P., and Palombini, F. (2018). Ephemeral Diffie-Hellman Over COSE (EDHOC). IETF Internet-Draft.

Selander, G., Mattsson, J. P., and Palombini, F. (2020). Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-00, Internet Engineering Task Force. Work in Progress.

Selander, G., Mattsson, J. P., and Palombini, F. (2022). Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-15, Internet Engineering Task Force. Work in Progress.

Xu, S., Zhao, Y., Ren, Z., Wu, L., Tong, Y., and Zhang, H. (2020). A symbolic model for systematically analyzing tee-based protocols. In Meng, W., Gollmann, D., Jensen, C. D., and Zhou, J., editors, *Information and Communications Security - 22nd International Conference, ICICS 2020, Copenhagen, Denmark, August 24-26, 2020, Proceedings*, volume 12282 of *Lecture Notes in Computer Science*, pages 126–144. Springer.