

# Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices

Karl Norrman<sup>1,2</sup>, Vaishnavi Sundararajan<sup>3</sup> and Alessandro Bruni<sup>4</sup>

<sup>1</sup>*KTH Royal Institute of Technology, Stockholm, Sweden*

<sup>2</sup>*Ericsson Research, Security, Stockholm, Sweden*

<sup>3</sup>*University of California Santa Cruz, USA*

<sup>4</sup>*IT University of Copenhagen, Copenhagen, Denmark*

*karl.norrman@ericsson.com, vasundar@ucsc.edu, brun@itu.dk*

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

**Abstract:** Constrained IoT devices are becoming ubiquitous in society and there is a need for secure communication protocols that respect the constraints under which these devices operate. EDHOC is an authenticated key establishment protocol for constrained IoT devices, currently being standardized by the Internet Engineering Task Force (IETF). A rudimentary version of EDHOC with only two key establishment methods was formally analyzed in 2018. Since then, the protocol has evolved significantly and several new key establishment methods have been added. In this paper, we present a formal analysis of all EDHOC methods in an enhanced symbolic Dolev-Yao model using the Tamarin tool. We show that not all methods satisfy the authentication notion injective of agreement, but that they all do satisfy a notion of implicit authentication, as well as Perfect Forward Secrecy (PFS) of the session key material. We identify other weaknesses to which we propose improvements. For example, a party may intend to establish a session key with a certain peer, but end up establishing it with another, trusted but compromised, peer. We communicated our findings and proposals to the IETF, which has incorporated some of these in newer versions of the standard.

## 1 INTRODUCTION

As IoT devices become more prevalent and get involved in progressively sensitive functions in society, the need to secure their communications becomes increasingly important. Most security analysis has focused on computationally strong devices, such as cars and web-cameras, where existing protocols like (D)TLS suffice. Constrained devices, on the other hand, which operate under severe bandwidth and energy consumption restrictions, have received much less attention. These devices may be simple sensors, which only relay environment measurements to a server every hour, but need to function autonomously without maintenance for long periods of time. The IETF standardized the Object Security for Constrained RESTful Environments (OSCORE) protocol to secure communications between constrained devices [Selander et al., 2019]. However, the OSCORE protocol requires a pre-established security context. The IETF has been discussing re-

quirements and mechanisms for a key exchange protocol, named Ephemeral Diffie-Hellman Over COSE (EDHOC), for establishing OSCORE security contexts. Naturally, EDHOC must work under the same constrained requirements as OSCORE itself. While not all use cases for EDHOC are firmly set, the overall goal is to establish an OSCORE security context, under message size limitations. It is therefore important to ensure that EDHOC satisfies fundamental security properties expected from a key exchange protocol.

The first incarnation of EDHOC appeared in March 2016. It contained two different key establishment methods, one based on a pre-shared Diffie-Hellman (DH) cryptographic core<sup>1</sup> and a second based on a variant of challenge-response signatures in the style of OPTLS [Krawczyk and Wee, 2016]. EDHOC is therefore a framework of several key establishment methods. In May 2018, the core based

---

<sup>1</sup>By a *cryptographic core*, or simply *core*, we mean an academic protocol, without encodings or application specific details required by an industrial protocol. A *key establishment method* is a core with some such details added.

<sup>a</sup> <https://orcid.org/0000-0003-0164-1478>

on challenge-response signatures was replaced by one based on SIGMA (SIGn-and-MAC) [Krawczyk, 2003, Selander et al., 2018]. Since then the protocol has undergone significant changes. Three new cores, mixing challenge-response signatures and regular signature for authentication, were added [Selander et al., 2020].

We formulate and formalize a security model covering all four key establishment methods, which is important especially since the specification [Selander et al., 2020] lacks a clear description of the intended security model and overall security goals.

We perform the analysis in a symbolic Dolev-Yao model. In this framework, we model messages as terms in an algebra, with operations such as encryption modelled as functions on these terms. These functions are assumed perfect, e.g., one cannot decrypt an encrypted message without access to the key. The adversary, while unable to break encryption or reverse hashing, is modelled as the network. That is, the adversary, can block reroute, replay and modify messages at will. A symbolic model like this, while slightly severe an abstraction, still allows us to analyze EDHOC for logical flaws without incurring the complexity of a computational model. The standardization process is ongoing, with the authors releasing newer versions of the specification (see Section 5 for more detail about how these versions differ from the one analyzed here).

## 1.1 Contributions

In this paper, we formally analyze the EDHOC protocol (with its four key establishment methods) using the Tamarin tool [Meier et al., 2013]. We present a formal model we constructed of the protocol as given in the specification [Selander et al., 2020].

We give an explicit adversary model for the protocol and verify properties such as session key material and entity authentication, and perfect forward secrecy, for all four methods.

The model itself is valuable as a basis for verifying further updates in the ongoing standardization. It is publicly available [Norrman et al., 2020]. It took several person-months to interpret the specification and construct the model. Termination requires a hand-crafted proof oracle to guide Tamarin.

We show that not all EDHOC's key establishment methods provide authentication according to the injective agreement definition on the session key material, and none on the initiator's identity. However, we show that all methods fulfill an implicit agreement property covering the session key material and the initiator's identity. We identify a number of subtleties,

ambiguities and weaknesses in the specification. For example, the authentication policy requirements allow situations where a party establishes session key material with a trusted but compromised peer, even though the intention was to establish it with a different trusted party. We provide remedies for the identified issues and have communicated these to the IETF and the specification authors, who have incorporated some of our suggestions and are currently considering how to deal with the remaining ones.

## 1.2 Comparison with Related Work

The May 2018 version of EDHOC was formally analyzed by [Bruni et al., 2018] using the ProVerif tool [Blanchet, 2001]. Their analysis covered a pre-shared key authenticated core and one based on SIGMA. The properties checked for therein were secrecy, PFS and integrity of application data, identity protection against an active adversary, and strong authentication.

In contrast to the key establishment methods analyzed by Bruni et al., which were based on the well-understood pre-shared key DH and SIGMA protocols, the three newly added methods combine two unilateral authentication protocols with the goal to constructing mutual authentication protocols. Combining two protocols, which individually provide unilateral authentication, is not guaranteed to result in a secure mutual authentication protocol [Krawczyk, 2016]. Consequently, even though the framework is similar to the one analyzed by Bruni et al., the cryptographic underpinnings have significantly increased in complexity, and is using mechanisms which have not previously been formally analyzed. The set of properties we check for is also different. Our analysis is further carried out using a different tool, namely Tamarin; different kinds of strategies to formulate and successfully analyze the protocol are required when working with this tool.

## 2 THE EDHOC PROTOCOL

We now present the structure of the protocol using notation for key material, elliptic curve operations and identities mostly adopted from NIST SP 800-56A Rev. 3 [Barker et al., 2018]. One notable difference is that we refer to the two roles executing the protocol as the initiator  $I$  and the responder  $R$ . We do this to avoid confusing roles with the parties taking them on. Values in the analysis are subscripted with  $I$  and  $R$  when necessary to distinguish which role is associated with the values.

## 2.1 Preliminaries

Private/public key pairs are written  $\langle d_{t,id}, Q_{t,id} \rangle$ , where  $d$  is the private key,  $Q$  the public key,  $t \in \{e, s\}$  denotes whether the key is ephemeral or static, and  $id$  is the role or party controlling the key pair. When irrelevant, we drop the subscript or parts thereof. Ephemeral key pairs are generated fresh for each instantiation of the protocol and static key pairs are long-term keys used for authentication. Static key pairs are suitable for either regular signatures or challenge-response signatures. When a party uses regular signatures for authentication, we say that they use the *signature based authentication method*, or SIG for short. When a party uses challenge-response signatures for authentication, we say that they use the *static key authentication method*, or STAT for short. The latter naming may appear confusing since signature keys are equally static, but is chosen to make the connection to the specification clear. We adopt the challenge-response terminology for this style of authentication from [Krawczyk, 2005].

EDHOC relies on COSE [Schaad, 2017] for elliptic curve operations and transforming points into bitstrings, and we therefore abstract those as follows. Signatures and verification thereof using party  $A$ 's key pair are denoted by  $sign_A(\cdot)$  and  $vf_A(\cdot)$  respectively. The DH-primitive combining a private key  $d$  and a point  $P$  is denoted by  $dh(d, P)$ . We abuse notation and let these function symbols denote operations on both points and the corresponding bitstrings.

## 2.2 Framework Structure

EDHOC's goal is to establish an OSCORE security context, including session key material denoted  $Z$ , and optionally transfer application data  $ad_1$ ,  $ad_2$  and  $ad_3$ . To accomplish this, the specification [Selander et al., 2020] gives a three-message protocol pattern, shown in Figure 1. We first describe this pattern and the parts that are common to all key establishment methods. Then we describe authentication and derivation of keys in more detail. The latter is what differentiates the key establishment methods from each other.

### 2.2.1 Protocol Pattern

The first two messages negotiate authentication methods  $M$  and a ciphersuite  $S_I$ . In  $M$ , the initiator  $I$  proposes which authentication method each party should use. These may differ, leading to four possible combinations: SIG-SIG, SIG-STAT, STAT-SIG and STAT-STAT. We refer to these *combinations* of authentication methods simply as *methods* to align with the specification terminology. The first authentication

method in a combination is the one proposed for the initiator and the last is the one proposed for the responder. The responder  $R$  may reject the choice of method or cipher suite with an error message, resulting in negotiation across multiple EDHOC sessions. Our analysis excludes error messages.

EDHOC's first two messages also exchange connection identifiers  $C_I$  and  $C_R$ , and public ephemeral keys,  $Q_{e,I}$  and  $Q_{e,R}$ . The connection identifiers  $C_I$  and  $C_R$ , described in Section 3.1 of the specification, deserve some elaboration. The specification describes these identifiers not as serving a security purpose for EDHOC, but only as aiding message routing to the correct EDHOC processing entity at a party. Despite this, the specification states that they may be used by OSCORE, or other protocols using the established security context, without restricting how they are to be used. Because EDHOC may need them in clear-text for routing, OSCORE cannot rely on them being secret. Section 7.1.1 of the specification requires the identifiers to be unique. Uniqueness is defined to mean that  $C_I \neq C_R$  for a given session and the specification requires parties to verify that this is the case. The same section also requires that OSCORE must be able to retrieve the security context based on these identifiers. The intended usage of  $C_I$  and  $C_R$  by OSCORE is not made specific and therefore it is not clear which properties should be verified. We verify that the parties agree on the established values.

The two last messages provide identification and authentication. Parties exchange identifiers for their long-term keys,  $ID_I$  and  $ID_R$ , as well as information elements,  $Auth_I$  and  $Auth_R$ , to authenticate that the parties control the corresponding long-term keys. The content of  $Auth_I$  and  $Auth_R$  depends on the authentication method associated with the corresponding long-term key. For example, if  $M = \text{SIG-STAT}$ , the responder  $R$  must either reject the offer or provide an  $ID_R$  corresponding to a key pair  $\langle d_{s,R}, Q_{s,R} \rangle$  suitable for use with challenge-response signatures, and compute  $Auth_R$  based on the static key authentication method STAT. In turn, the initiator  $I$  must provide an  $ID_I$  corresponding to a key pair  $\langle d_{s,I}, Q_{s,I} \rangle$  suitable for a regular signature, and compute  $Auth_I$  based on the signature based authentication method SIG.

### 2.2.2 Authentication

Regardless of whether STAT or SIG is used to compute  $Auth_R$ , a MAC is first computed over  $ID_R$ ,  $Q_{s,R}$ , a transcript hash of the information exchanged so far, and  $ad_2$  if included. The MAC is the result of encrypting the empty string with the Authenticated Encryption with Additional Data (AEAD) algorithm from the ciphersuite  $S_I$ , using the mentioned infor-

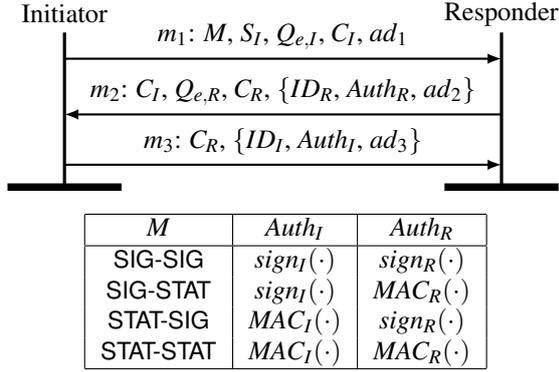


Figure 1: Structure of EDHOC:  $\{t\}$  means  $t$  is encrypted and integrity protected.

mation as additional data. The MAC key is derived from the ephemeral key material  $Q_{e,I}$ ,  $Q_{e,R}$ ,  $d_{e,I}$  and  $d_{e,R}$ , where  $I$  computes  $dh(d_{e,I}, Q_{e,R})$  and  $R$  computes  $dh(d_{e,R}, Q_{e,I})$ , both resulting in the same output in the usual DH way.

In case  $R$  uses the SIG authentication method,  $Auth_R$  is  $R$ 's signature over the MAC itself and the data that the MAC already covers. In case  $R$  uses the STAT authentication method,  $Auth_R$  is simply the MAC itself. However, when using STAT, the key for the MAC is derived, not only from the ephemeral key material, but also from  $R$ 's long-term key  $\langle d_{s,R}, Q_{s,R} \rangle$ . For those familiar with OPTLS, this corresponds to the 1-RTT semi-static pattern computing the MAC key  $sfk$  for the  $sfm$  message [Krawczyk and Wee, 2016]. The content of  $Auth_I$  is computed in the corresponding way for the initiator  $I$ . In Figure 1, we denote a MAC using a key derived from both  $\langle d_{s,R}, Q_{s,R} \rangle$  and  $\langle d_{e,I}, Q_{e,I} \rangle$  by  $MAC_I$ , and a MAC using a key derived from both  $\langle d_{s,I}, Q_{s,I} \rangle$  and  $\langle d_{e,R}, Q_{e,R} \rangle$  by  $MAC_R$ .

Parts of the last two messages are encrypted and integrity protected, as indicated in 1. The second message is encrypted by XORing the output of the key derivation function HKDF (see Section 2.2.3) on to the plain text. The third message is encrypted and integrity protected by the AEAD algorithm determined by the ciphersuite  $S_I$ .

### 2.2.3 Key Schedule

At the heart of EDHOC is the key schedule depicted in Figure 2. EDHOC uses two functions from the HKDF interface [Krawczyk and Eronen, 2010] to derive keys. HKDF-extract constructs uniformly distributed key material from random input and a salt, while HKDF-expand generates keys from key material and a salt.

The key schedule is rooted in the ephemeral DH key  $P_e$ , which is computed as  $dh(d_{e,I}, Q_{e,R})$  by  $I$  and as

$dh(d_{e,R}, Q_{e,I})$  by  $R$ . From  $P_e$ , three intermediate keys  $PRK_{2e}$ ,  $PRK_{3e2m}$  and  $PRK_{4x3m}$  are derived during the course of protocol execution. Each of them is used for a specific message in the protocol, and from these intermediate keys, encryption and integrity keys ( $K_{2e}$ ,  $K_{2m}$ ,  $K_{3ae}$ , and  $K_{3m}$ ) for that message are derived. The salt for generating  $PRK_{2e}$  is the empty string.

The protocol uses a running transcript hash  $th$ , which includes the information transmitted so far. The value of the hash, denoted  $th_i$  for the  $i$ th message, is included in key derivations as shown in Figure 2.

Successful protocol execution establishes the session key material  $Z$  for OSCORE.  $Z$  can be considered a set that always includes  $P_e$ . If the initiator uses the STAT authentication method,  $Z$  also includes  $dh(d_{e,R}, Q_{s,I}) = dh(d_{s,I}, Q_{e,R})$ , which we denote by  $P_I$ . If the responder uses the STAT authentication method, it also includes  $dh(d_{e,I}, Q_{s,R}) = dh(d_{s,R}, Q_{e,I})$ , which we denote by  $P_R$ . From the session key material, a key exporter (EDHOC-Exporter) based on HKDF is used to extract keys required for OSCORE.

As an illustrative example of the entire process, we refer to Figure 3, which depicts the protocol pattern, operations and key derivations for the SIG-STAT method in more detail.

## 3 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.

### 3.1 Adversary Model

We verify EDHOC in the symbolic Dolev-Yao model, with idealized cryptographic primitives, e.g. encrypted messages can only be decrypted using the key, no hash collisions exist etc. The adversary controls the communication channel, and can interact with an unbounded number of sessions of the protocol, dropping, injecting and modifying messages to their liking.

In addition to the basic Dolev-Yao model, we also consider two more adversary capabilities, namely long-term key reveal and ephemeral key reveal. Long-term key reveal models the adversary compromising a

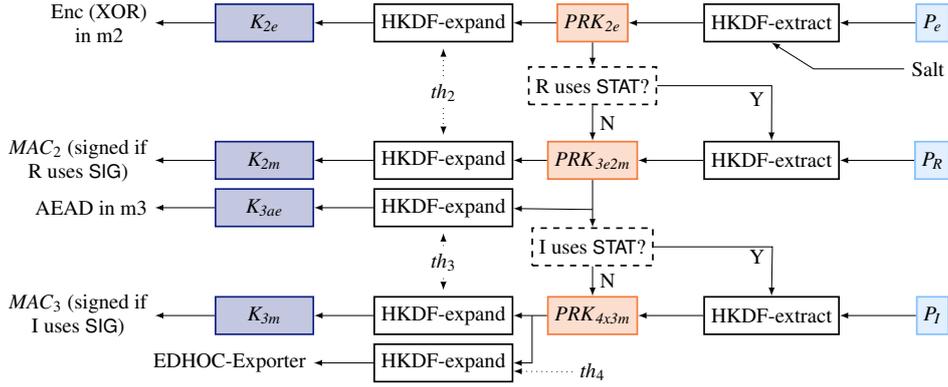


Figure 2: Key schedule: Light blue boxes hold DH keys ( $P_e, P_I, P_R$ ), orange boxes intermediate key material ( $PRK_{2e}, PRK_{3e2m}, PRK_{4x3m}$ ), and dark blue boxes keys for AEAD or XOR encryption ( $K_{2e}, K_{2m}, K_{3ae}, K_{3m}$ ). Dashed boxes are conditionals.

party  $A$ 's long-term private key  $d_{s,A}$  at time  $t$ , and we denote this event by  $A_{\text{LTK}}^t(A)$ . The event  $A_{\text{Eph}}^t(A, k)$  represents that the adversary learns the ephemeral private key  $d_{e,A}$  used by party  $A$  at time  $t$  in a session establishing key material  $k$ . These two capabilities model the possibility to store and operate on long-term keys in a secure module, whereas ephemeral keys may be stored in a less secure part of a device. This is more granular and realistic than assuming that the adversary has equal opportunity to access both types of keys.

We now define and formalize the security properties we are interested in, and then describe how we encode them into Tamarin. The adversary model becomes part of the security properties themselves.

### 3.2 Formalization of Properties

We use the Tamarin verification tool [Meier et al., 2013] to encode the model and verify properties. This tool uses a fragment of temporal first order logic to reason about events and knowledge of the parties and of the adversary. For conciseness we use a slightly different syntax than that used by Tamarin, but which has a direct mapping to Tamarin's logic.

Event types are predicates over global states of system execution. Let  $E$  be an event type and let  $t$  be a timestamp associated with a point in a trace. Then  $E^t(p_i)_{i \in \mathbb{N}}$  denotes an event of type  $E$  associated with a sequence of parameters  $(p_i)_{i \in \mathbb{N}}$  at time  $t$  in that trace. In general, more than one event may have the same timestamp and hence timestamps form a quasi order, which we denote by  $t_1 < t_2$  when  $t_1$  is before  $t_2$  in a trace. We define  $\doteq$  analogously. However, two events of the same type cannot have the same timestamp, so  $t_1 \doteq t_2$  implies  $E^{t_1} = E^{t_2}$ . Two events having the same timestamp does not imply the there is a fork in the

trace, only that the two events happen simultaneously. This notation corresponds to Tamarin's use of action facts  $E(p_i)_{i \in \mathbb{N}}@t$ .

The event  $\mathcal{K}^t(p)$  denotes that the adversary knows a parameter  $p$  at time  $t$ . Parameters are terms in a term algebra of protocol specific operations and generic operations, e.g., tuples  $\langle \cdot \rangle$ . Intuitively,  $\mathcal{K}^t(p)$  evaluates to true when  $p$  is in the closure of the parameters the adversary observed by interacting with parties using the protocol, under the Dolev-Yao message deduction operations and the advanced adversary capabilities up until time  $t$ . For a more precise definition of knowledge management, we refer to [Meier et al., 2013]. An example of a formula is

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

expressing that if there is a time  $t$  when the adversary knows the tuple  $\langle k, k' \rangle$ , then the adversary knows both  $k$  and  $k'$  at the same point in time.

An initiator  $I$  considers the protocol run started when it sends a message  $m_1$  (event type  $\mathbf{I}_S$ ) and the run completed after sending a message  $m_3$  (event type  $\mathbf{I}_C$ ). Similarly, a responder  $R$  considers the run started upon receiving a  $m_1$  (event type  $\mathbf{R}_S$ ), and completed upon receiving a  $m_3$  (event type  $\mathbf{R}_C$ ).

#### 3.2.1 Perfect Forward Secrecy (PFS)

Informally, PFS captures the idea that session key material remains secret even if a long-term key leaks in the future. We define PFS for session key material  $Z$  as **PFS** in Figure 4.

The first parameter  $I$  of the  $\mathbf{I}_C$  event represents the initiator's identity, and the second,  $R$ , represents that  $I$  believes  $R$  to be playing the responder role. The third parameter,  $Z$ , is the established session key material. The parameters of the  $\mathbf{R}_C$  event are defined analogously. Specifically, the first parameter of  $\mathbf{R}_C$



$$\begin{aligned}
\mathbf{PFS} &\triangleq \forall I, R, Z, t_2, t_3. \mathcal{K}^{t_3}(Z) \wedge (\mathbf{I}_C^{t_2}(I, R, Z) \vee \mathbf{R}_C^{t_2}(I, R, Z)) \rightarrow \\
&\quad (\exists t_1. \mathbf{A}_{\text{LTK}}^{t_1}(I) \wedge t_1 \leq t_2) \vee (\exists t_1. \mathbf{A}_{\text{LTK}}^{t_1}(R) \wedge t_1 \leq t_2) \vee (\exists t_1. \mathbf{A}_{\text{Eph}}^{t_1}(R, Z)) \vee (\exists t_1. \mathbf{A}_{\text{Eph}}^{t_1}(I, Z)) \\
\mathbf{InjAgree}_I &\triangleq \forall I, R, Z, S, t_2. \mathbf{I}_C^{t_2}(I, R, Z, S) \rightarrow \\
&\quad (\exists t_1. \mathbf{R}_S^{t_1}(R, Z, S) \wedge t_1 \leq t_2) \wedge (\forall I' R' t'_1. \mathbf{I}_C^{t'_1}(I', R', Z, S) \rightarrow t'_1 \doteq t_1) \vee (\exists t_1. \mathbf{A}_{\text{LTK}}^{t_1}(R) \wedge t_1 \leq t_2) \\
\mathbf{InjAgree}_R &\triangleq \forall I, R, Z, S, t_2. \mathbf{R}_C^{t_2}(I, R, Z, S) \rightarrow \\
&\quad (\exists t_1. \mathbf{I}_S^{t_1}(I, R, Z, S) \wedge t_1 \leq t_2) \wedge (\forall I' R' t'_1. \mathbf{R}_C^{t'_1}(I', R', Z, S) \rightarrow t'_1 \doteq t_1) \vee (\exists t_1. \mathbf{A}_{\text{LTK}}^{t_1}(I) \wedge t_1 \leq t_2). \\
\mathbf{ImpAgree}_I &\triangleq \forall I, R, Z, S, t_1. \mathbf{I}_C^{t_1}(I, R, Z, S) \rightarrow \\
&\quad (\forall I', R', S', t_2. \mathbf{R}_C^{t_2}(I', R', Z, S') \rightarrow (I = I' \wedge R = R' \wedge S = S')) \\
&\quad \wedge (\forall I', R', S', t'_1. (\mathbf{I}_C^{t'_1}(I', R', Z, S') \rightarrow t'_1 \doteq t_1) \\
&\quad \vee (\exists t_0. \mathbf{A}_{\text{LTK}}^{t_0}(R) \wedge t_0 \leq t_1) \vee (\exists t_0. \mathbf{A}_{\text{Eph}}^{t_0}(R, Z)) \vee (\exists t_0. \mathbf{A}_{\text{Eph}}^{t_0}(I, Z)).
\end{aligned}$$

Figure 4: Formalization of security properties and adversary model.

methods a form of *implicit agreement* on all the parameters mentioned above. We take inspiration from the computational model definitions of implicit authentication, proposed by [Delpech de Saint Guilhem et al., 2020], to modify classical injective agreement into an implicit property. A small but important difference between our definition and theirs, is that they focus on authenticating a key and related identities, whereas we extend the more general concept of agreeing on a set of parameters, starting from the idea of injective agreement [Lowe, 1997]. We use the term *implicit* in this context to denote that a party  $A$  assumes that any other party  $B$  who knows the session key material  $Z$  must be the intended party, and that  $B$  (if honest) will also agree on a set  $S$  of parameters computed by the protocol, one of which is  $Z$ . When implicit agreement holds for both roles, upon completion,  $A$  is guaranteed that  $A$  has been or is engaged in exactly one protocol run with  $B$  in the opposite role, and that  $B$  has been or will be able to agree on  $S$ . The main difference from injective agreement is that  $A$  concludes that if  $A$  sends the last message and this reaches  $B$ , then  $A$  and  $B$  have agreed on  $I$ ,  $R$  and  $S$ . While almost full explicit key authentication, as defined by [Delpech de Saint Guilhem et al., 2020], is a similar property, our definition does not require key confirmation, so our definition is closer to their definition of implicit authentication. In the Tamarin model we split the property into one lemma for  $I$  ( $\mathbf{ImpAgree}_I$ ) and one for  $R$  ( $\mathbf{ImpAgree}_R$ ) to save memory during verification. We show only the definition for  $I$  in Figure 4, because it is symmetric to the

one for  $R$ .

For implicit agreement to hold for the initiator  $I$ , the ephemeral keys can never be revealed. Intuitively, the reason for this is that the implicit agreement relies on that whomever knows the session key material is the intended responder. An adversary with access to the ephemeral keys and the public keys of both parties can compute the session key material produced by all methods. However, the responder  $R$ 's long-term key can be revealed after  $I$  completes its run, because the adversary is still unable to compute  $P_e$ . The initiator's long-term key can also be revealed at any time without affecting  $I$ 's guarantee for the same reason.

### 3.2.3 Agreed Parameters

The initiator  $I$  gets injective and implicit agreement guarantees on the following partial set  $S_P$  of parameters:

- the roles played by itself and its peer,
- responder identity,
- session key material (which varies depending on EDHOC method),
- context identifiers  $C_I$  and  $C_R$ , and
- cipher suites  $S_I$ .

Because EDHOC aims to provide identity protection for  $I$ , there is no injective agreement guarantee for  $I$  that  $R$  agrees on the initiator's identity. For the same reason, there is no such guarantee for  $I$  with respect to the  $P_I$  part of the session key material when  $I$  uses the STAT authentication method. There is, however,

an implicit agreement guarantee for  $I$  that  $R$  agrees on  $I$ 's identity and the full session key material. Since  $R$  completes after  $I$ ,  $R$  can get injective agreement guarantees on more parameters, namely also the initiator's identity and the full session key material for all methods. 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.

### 3.2.4 Inferred Properties

From the above, other properties can be inferred to hold in our adversary model. Protocols where a party does not get confirmation that their peer knows the session key material may be susceptible to *Key-Compromise Impersonation (KCI)* attacks [Blake-Wilson et al., 1997]. Attacks in this class allow an adversary in possession of a party  $A$ 's secret long-term key to coerce  $A$  to complete a protocol run believing it authenticated a certain peer  $B$ , but where  $B$  did not engage with  $A$  at all in a run. Because both our above notions of agreement ensure agreement on identities, roles and session key material, all methods passing verification of those are also resistant to KCI attacks.

If a party  $A$  can be coerced into believing it completed a run with  $B$ , but where the session key material is actually shared with  $C$  instead, the protocol is vulnerable to an *Unknown Key-Share (UKS)* attack [Blake-Wilson et al., 1997]. For the same reason as for KCI, any method for which our agreement properties hold is also resistant to UKS attacks.

From the injective agreement properties it follows that each party is assured the identity of its peer upon completion. Therefore, the agreement properties also capture *entity authentication*.

## 3.3 Tamarin

We chose Tamarin to model and verify EDHOC in the symbolic model. It is an interactive verification tool in which models are specified as multi-set rewrite rules that define a transition relation. The elements of the multi-sets are facts representing the global system state. Rules are equipped with event annotations called actions. Sequences of actions make up execution traces, over which logic formulas are defined.

Multi-set rewrite rules are written  $l \dashv[e] \rightarrow r$ , where  $l$  and  $r$  are multi-sets of facts, and  $e$  is a multi-set of actions. Facts and actions are  $n$ -ary predicates over a term algebra, which defines a set of function symbols, variables and names. Tamarin checks equality of these terms under an equational theory  $E$ . For example, one can write  $dec(enc(x,y),y) =_E x$  to denote that symmetric decryption reverses the encryption operation under  $E$ . The equational theory  $E$  is

fixed per model, and hence we omit the subscript. Tamarin supports let-bindings and tuples as syntactic sugar to simplify model definitions. It also provides built-in rules for Dolev-Yao adversaries and for managing their knowledge. We implement events using actions, and parameters associated with events using terms of the algebra.

### 3.3.1 Protocol Rules and Equations

Tamarin allows users to define new function symbols, equational theories and rules, which are added to the set of considered rules during verification. For example, in our model we have a symbol to denote authenticated encryption, for which Tamarin produces a rule of the following form:

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

to denote that if the adversary knows a key  $k$ , a message  $m$ , the authenticated data  $ad$ , and an algorithm  $ai$ , then they can construct the encryption, and thus get to know the message  $aeadEncrypt(k, m, ad, ai)$ .

## 3.4 Tamarin Encoding of EDHOC

We model all four methods of EDHOC, namely SIG-SIG, SIG-STAT, STAT-SIG and STAT-STAT. Because the methods share a lot of common structure, we derive their Tamarin-models from a single specification written with the aid of the M4 macro language. To keep the presentation brief, we only present the STAT-SIG method, as it illustrates the use of two different asymmetric authentication methods simultaneously. The full Tamarin code for all models can be found at [Norman et al., 2020]. Variable names used in the code excerpts here are sometimes shortened compared to the model itself to fit the paper format.

### 3.4.1 Primitive Operations

Our model uses the built-in theories of exclusive-or and DH operations, as in [Dreier et al., 2018, Schmidt et al., 2012]. Hashing is modeled via the built-in hashing function symbol augmented with a public constant as additional input, modelling different hash functions. The HKDF interface is represented by `expa` for the expansion operation and `extr` for the extraction operation. Signatures use Tamarin's built-in theory for `sign` and `verify` operations. For AEAD operations on key  $k$ , message  $m$ , additional data  $ad$  and algorithm identifier  $ai$ , we use `aeadEncrypt(m, k, ad, ai)` for encryption. Decryption with verification of the integrity is defined via the equation

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

The integrity protection of AEAD covers  $ad$ , and this equation hence requires an adversary to know  $ad$  even if they only wish to decrypt the data. To enable the adversary to decrypt without needing to verify the integrity we add the equation

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

The latter equation is not used by honest parties.

### 3.4.2 Protocol Environment and Adversary Model

We model the binding between a party’s identity and their long-term key pairs using rules for SIG- and STAT-based methods separately.

```
rule registerLTK_SIG:
  [Fr(~ltk)]--[UniqLTK($A, ~ltk)]->
    [!LTK_SIG($A, ~ltk),
     !PK_SIG($A, pk(~ltk)),
     Out(<$A, pk(~ltk)>)]
rule registerLTK_STAT:
  [Fr(~ltk)]--[UniqLTK($A, ~ltk)]->
    [!LTK_STAT($A, ~ltk),
     !PK_STAT($A, 'g'~ltk),
     Out(<$A, 'g'~ltk>)]
```

The fact  $Fr(\sim ltk)$  creates a fresh term  $ltk$ , representing a long-term secret key, not known to the adversary. The fact  $Out(\langle \$A, pk(\sim ltk) \rangle)$  sends the identity of the party owning the long-term key and the corresponding public key to the adversary. The event  $UniqLTK$  together with a corresponding restriction models the fact that the each party is associated with exactly one long-term key. Consequently, an adversary cannot register additional long-term keys for an identity. In line with the EDHOC specification, this models an external mechanism ensuring that long term keys are bound to correct identities, e.g., a certificate authority.

We rely on Tamarin’s built-in message deduction rules for a Dolev-Yao adversary. To model an adversary compromising long-term keys, i.e., events of type  $A_{LTK}$ , and revealing ephemeral keys, i.e., events of type  $A_{Eph}$ , we use standard reveal rules. The timing of reveals as modelled by these events is important. The long-term keys can be revealed on registration, before protocol execution. The ephemeral key of a party can be revealed when the party completes, i.e., at events of type  $I_C$  and  $R_C$ .<sup>2</sup>

### 3.4.3 Protocol Roles

We model each method of the protocol with four rules:  $I1$ ,  $R2$ ,  $I3$  and  $R4$  (with the method suffixed

<sup>2</sup>A stronger, and perhaps more realistic, model would reveal ephemeral keys upon creation at the start of the run, but we failed to get Tamarin to terminate on this.

to the rule name). Each of these represent one step of the protocol as run by the initiator  $I$  or the responder  $R$ . The rules correspond to the event types  $I_S$ ,  $R_S$ ,  $I_C$ , and  $R_C$ , respectively. Facts prefixed with  $StI$  carry state information between  $I1$  and  $I3$ . A term unique to the current thread,  $tid$ , links two rules to a given state fact. Similarly, facts prefixed with  $StR$  carry state information between the responder role’s rules. Line 28 in the  $R2\_STAT\_SIG$  rule shown below illustrate one such use of state facts.

We do not model the error message that  $R$  can send in response to message  $m_1$ , and hence our model does not capture the possibility for  $R$  to reject  $I$ ’s offer.

We model the XOR encryption of  $CIPHERTEXT\_2$  with the key  $K\_2e$  using Tamarin’s built in theory for XOR, and allow each term of the encrypted element to be attacked individually. That is, we first expand  $K\_2e$  to as many key-stream terms as there are terms in the plaintext tuple by applying the HKDF-expand function to unique inputs per term. We then XOR each term in the plaintext with its own key-stream term. This models the specification closer than if we would have XORed  $K\_2e$  as a single term onto the plaintext tuple. The XOR encryption can be seen in lines 19–22 in the listing of  $R2\_STAT\_SIG$  below.

```
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'>,
11            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,
17             $cAEAD0)
17  authV = sign(<assocData2, MAC_2>, ~ltk)
18  plainText2 = <$V, authV>
19  K_2e = expa(<$cAEAD0, TH_2,
20            'K_2e'>, prk_2e)
20  K_2e_1 = expa(<$cAEAD0, TH_2,
21            'K_2e', '1'>, prk_2e)
21  K_2e_2 = expa(<$cAEAD0, TH_2,
22            'K_2e', '2'>, prk_2e)
22  CIPHERTEXT_2 = <$V XOR K_2e_1,
23                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),
27   In(m1), Fr(~CR), Fr(~yy), Fr(~tid)]
27  --[ExpRunningR(~tid, $V, exp_sk, agreed),
28     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)]

```

To implement events and to bind them to parameters, we use actions. For example, the action `ExpRunningR(~tid, $V, exp_sk, agreed)` in line 27 above implements binding of an event of type  $\mathbf{R}_S$  to the parameters and session key material.

As explained in Section 3.2.3, it is not possible to show injective agreement on session key material when it includes  $P_I$  (not visible in the rule  $\mathbf{R2\_STAT\_SIG}$ ). Therefore, we use certain actions to implement events that include  $P_I$  in the session key material and other actions that do not. Session key material which includes (resp. does not include)  $P_I$  is referred to as `imp_sk` (resp. `exp_sk`) in the Tamarin model. In the case of  $\mathbf{SIG\_SIG}$  and  $\mathbf{SIG\_STAT}$ , therefore, `imp_sk` is the same as `exp_sk`.

### 3.5 Tamarin Encoding of Properties

The properties and adversary model we defined in Section 3.2 translate directly into Tamarin’s logic, using the straightforward mapping of events to the actions emitted from the model. As an example, we show the lemma for verifying the property  $\mathbf{PFS}$ .

```

1 lemma secrecyPFS:
2 all-traces
3 "All u v sk #t3 #t2.
4 (K(sk)@t3 & CompletedRun(u, v, sk)@t2) ==>
5 ( (Ex #t1. LTKRev(u)@t1 & #t1 < #t2)
6 | (Ex #t1. LTKRev(v)@t1 & #t1 < #t2)
7 | (Ex #t1. EphKeyRev(sk)@t1) )"

```

The action `CompletedRun(u, v, sk)` in line 4 is emitted by both the rules  $\mathbf{I3}$  and  $\mathbf{R4}$ , and corresponds to the disjunction of events  $\mathbf{I}_C^2 \vee \mathbf{R}_C^2$  in the definition of  $\mathbf{PFS}$  in Section 3.2.1. Similarly, `EphKeyRev(sk)` in line 7 models that the ephemeral key is revealed for either  $I$  or  $R$ , or both.

## 4 DISCUSSION

There are a few places where EDHOC can be improved, which we found during this work and communicated to the authors. We discuss them below.

### 4.1 Unclear Intended Use

The EDHOC specification lists several security goals, but they are imprecise and difficult to interpret due to lack of context and intended usage descriptions. Without knowing how the protocol is to be used, it is not clear whether the listed security goals are the most important ones for constrained IoT devices.

The abstract goal of EDHOC is simple: establish an OSCORE security context using few roundtrips and small messages. From that, the design of EDHOC is mainly driven by what can be achieved given the technical restrictions. Focusing too much on what can be achieved within given restrictions, and paying too little attention to the use cases where the protocol is to be used and their specific goals, risks resulting in sub-optimal trade-offs and design decisions.

EDHOC is intended to cover a variety of use cases, many of which are difficult to predict today. However, this does not prevent collecting *typical* use cases and user stories to identify more specific security goals that will be important in most cases.

While constructing our model, we made up simple user stories to identify security properties of interest. Several of these revealed subtleties and undefined aspects of EDHOC. We informed the EDHOC authors, who addressed these aspects in the specification.

#### 4.1.1 (Non-)Repudiation

An access control solution for a nuclear power-plant may need to log who is passing through a door, whereas it may be undesirable for, say, a coffee machine to log a list of users along with their coffee preferences. Via this simple thought experiment, we realized that the specification did not consider the concept of (non)-repudiation. In response, the authors of the specification added a paragraph discussing how different methods relate to (non)-repudiation.

#### 4.1.2 Unintended Peer Authentication

According to Section 3.2 of the specification, parties must be configured with a policy restricting the set of peers they run EDHOC with. However, the initiator is not required to verify that the  $ID_R$  received in the second message is the same as the one intended. The following attack scenario is therefore possible.

Suppose someone has configured all devices in their home to be in the allowed set of devices, but that one of the devices ( $A$ ) is compromised. If another device  $B$ , initiates a connection to a third device  $C$ , the compromised device  $A$  may interfere by responding in  $C$ ’s place, blocking the legitimate response from  $C$ . Since  $B$  does not verify that the identity indicated in the second message matches the intended identity  $C$ , and device  $A$  is part of the allowed set,  $B$  will complete and accept the EDHOC run with device  $A$  instead of the intended  $C$ . The obvious solution is for the initiator to match  $ID_R$  to the intended identity indicated by the application, which we included in our model. We have communicated this to the EDHOC authors and they are considering a resolution.

## 4.2 Unclear Security Model

We argue that the specification gives too little information about what capabilities an adversary is assumed to have, and that this leads to unclear design goals and potentially sub-optimal design.

Even though EDHOC incorporates cryptographic cores from different academic security protocols, its design does not take into account the adversary models for which these protocols were designed. For example, OPTLS, whose cryptographic core is essentially the same as the STAT authentication method, is designed to be secure in the CK model [Canetti and Krawczyk, 2002]. The CK security model explicitly separates the secure storage of long-term keys from storage of session state and ephemeral keys. This is appropriate for modelling the use of secure modules.

The EDHOC authors indicated to us that it was not necessary to consider compromised ephemeral keys separately from compromised long-term keys. The rationale is that SIGMA cannot protect against compromised ephemeral keys [EDHOC authors, 2020]. That rationale is presumably based on the fact that the SIG-SIG method is closely modeled on the SIGMA-I variant of SIGMA, and that it would be preferable to obtain a homogeneous security level among the EDHOC methods. That is only true, however, when restricting attention to session key confidentiality of an ongoing session. Secure modules provide value in other ways, for example, by allowing constructions with Post-Compromise Security (PCS) guarantees. We discussed this with the authors, and the latest version of the specification [Selander et al., 2021] includes recommendations on storage of long-term keys and operations on these inside a secure module.

## 4.3 Session Key Material

EDHOC establishes session key material, from which session keys can be derived using the EDHOC-Exporter. The session key material is affected by  $P_e$ , and if a party uses the STAT authentication method, also by that party’s secret static long-term key. As shown in Section 3, mutual injective agreement cannot be achieved for  $P_I$ . If this property is not important for constrained IoT devices which cannot use any of the other methods, then one can simply accept that the methods have different authentication strengths. Otherwise, this is a problem.

We identified three alternatives for resolving this. One alternative is to include  $ID_I$ , or its hash, in the first and second messages. This would, however, increase message sizes and prevent initiator identity protection, which are grave concerns for EDHOC. A

second alternative is to not derive the session key material from  $P_I$ . Doing so, however, deviates from the design of OPTLS (and similar protocols from which the STAT-based methods are derived), where the inclusion of  $P_I$  plays a crucial part in the security proof of resistance against initiator ephemeral key compromise. The third alternative is to include a fourth message from responder to initiator, carrying a MAC based on a key derived from session key material including  $P_I$ . Successful MAC verification guarantees to the initiator that the responder injectively agrees on  $P_I$ . We presented the options to IETF, and they decided to add a fourth message as an option in the latest version of the specification [Selander et al., 2021].

We verified that all methods enjoy a common, but weaker, property: mutual implicit agreement on all of  $P_e, P_I$  and  $P_R$ , where applicable.

## 5 CONCLUSIONS AND FUTURE WORK

We formally modeled all four methods of the EDHOC specification using Tamarin. We formulated several important security properties and identified precise adversary models in which we verified these. The properties are shown in Table 1. Mutual injective agreement covers the set of parameters  $S_P$ : responder identity, roles, session key material (except for  $P_I$  when initiator uses the STAT authentication method), context identifiers  $C_I$  and  $C_R$ , and cipher suites  $S_I$ . The responder in addition is ensured agreement on the initiators identity and  $P_I$ , i.e., on the set  $S_F$ . Implicit agreement covers all previously mentioned parameters for both peers. Verification of all lemmas, including model validation lemmas, took 42 minutes on an Intel Core i7-6500U 2.5GHz using two cores. 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 verified each method in isolation. Verifying security under composition is left as future work.

In this work, we have analyzed the EDHOC version as of July 2020 [Selander et al., 2020]. There are newer versions, with the most recent version as of February 2021 [Selander et al., 2021]. However, the changes to the protocol over these versions are not particularly significant for our analysis.

Table 1: 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  $P_I$ .

	SIG-SIG	SIG-STAT	STAT-SIG	STAT-STAT
Injective agreement for I	$S_F$	$S_F$	$S_P$	$S_P$
Injective agreement for R	$S_F$	$S_F$	$S_F$	$S_F$
Implicit agreement for I	$S_F$	$S_F$	$S_F$	$S_F$
Implicit agreement for R	$S_F$	$S_F$	$S_F$	$S_F$
PFS for session key material	✓	✓	✓	✓

## 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.

## REFERENCES

- Barker, E., Chen, L., Roginsky, A., Vassilev, A., and Davis, R. (2018). SP 800-56A Rev. 3: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. Technical report, NIST.
- 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.
- Canetti, R. and Krawczyk, H. (2002). Security analysis of IKE’s signature-based key-exchange protocol. In *Proc. of CRYPTO*, pages 143–161.
- 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.
- 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.
- 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. (2016). A unilateral-to-mutual authentication compiler for key exchange (with applications to client authentication in TLS 1.3). In *Proc. of ACM CCS*, pages 1438–1450.
- 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.
- Norrman, K., Sundararajan, V., and Bruni, A. (2020). EDHOC model. <https://github.com/hoheinzollern/EDHOC-Verification/tree/master/models/edhoc/secrypt21-tamarin.tgz>.
- Schaad, J. (2017). CBOR Object Signing and Encryption (COSE). RFC 8152.
- 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., and Palombini, F. (2020). Ephemeral Diffie-Hellman Over COSE (EDHOC). IETF Internet-Draft draft-selander-lake-edhoc-00. <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-00>.
- Selander, G., Mattsson, J., Palombini, F., and Seitz, L. (2019). Object Security for Constrained RESTful Environments (OSCORE). RFC 8613.
- Selander, G., Mattsson, J. P., and Palombini, F. (2018). Ephemeral Diffie-Hellman Over COSE (EDHOC). IETF Internet-Draft draft-selander-ace-cose-ecdhe-08. <https://datatracker.ietf.org/doc/html/draft-selander-ace-cose-ecdhe-08>.
- Selander, G., Mattsson, J. P., and Palombini, F. (2021). Ephemeral Diffie-Hellman Over COSE (EDHOC). IETF Internet-Draft draft-ietf-lake-edhoc-05. <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-05>.