Formal Verification of Ephemeral Diffie-Hellman Over COSE (EDHOC)

Alessandro Bruni, Thorvald Sahl Jørgensen,
Theis Grønbech Petersen, Carsten Schürmann [⋆]

IT University of Copenhagen, Copenhagen, Denmark

Abstract. Ephemeral Diffie-Hellman over COSE (EDHOC) [1] is an authentication protocol that aims to replace TLS for resource constrained Internet of Things (IoT) devices using a selection of lightweight ciphers and formats. It is inspired by the newest Internet Draft of TLS 1.3 [2] and includes reduced round-trip modes. Unlike TLS 1.3, EDHOC is designed from scratch, and does not have to support legacy versions of the protocol. As the protocol is neither well-known nor has been used in practice it has not been scrutinized to the extent it should be.
The objective of this paper is to verify security properties of the protocol, including integrity, secrecy, and perfect forward secrecy properties. We use ProVerif [3] to analyze these properties formally. We describe violations of specific security properties for the reduced round-trip modes. The flaws were reported to the authors of the EDHOC protocol.

1 Introduction

Ephemeral Diffie-Hellman over COSE (EDHOC [1]) is a lightweight authenticated key exchange protocol proposed by Selander et al., part of a family of protocols for Internet of Things (IoT) devices, and intended to be used in conjunction with OSCORE [4]. COSE refers to a proposed IETF Encryption standard [12]. The rationale for designing EDHOC is to provide a lightweight alternative to the TLS handshake standard that can fit in a micro-controller. For this purpose the backwards compatibility features of TLS are an unnecessary obstacle to an efficient implementation. Furthermore, a clean-slate design can help improve performance and also security, by supporting by default Perfect Forward Secrecy (PFS) and following the best practices of the SIGn-and-MAC (SIGMA) family of protocols [5], offering two modes of operation: asymmetric using public identities, and symmetric using both public identities and pre-shared keys.

With this work we provide a formal analysis of the EDHOC protocol, which is currently an Internet Draft on track for standardisation. We analyze the draft

[⋆] This work was funded in part through the Danish Council for Strategic Research, Programme Comission on Strategic Growth Technologies under grant 10-092309. This publication was also made possible by NPRP grant NPRP 7-988-1-178 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

version 08 of the specification [1] and uncover hidden assumptions in the use of encrypted payloads in the protocol. In particular, it is possible for an attacker to learn the content of encrypted application data during key exchange in the asymmetric version of the protocol, and to violate perfect forward secrecy of application data for protocol executions in both modes (symmetric and the asymmetric) when the attacker actively interferes with the session establishment. These findings, albeit similar to other instances of SIGMA key establishment protocols [6], are important to prevent potential misuse of the protocol features, and have been incorporated into later revisions of the draft.

Compared with other authentication protocols, for example TLS 1.3 [2], EDHOC is conceptually much simpler, because it implements only the handshake part without certificate handling and it has to run reliably on low energy devices. EDHOC optimizes the number of messages to be exchanged, the length of the message, and the number of encryption, decryption and signing operations.

However, the low complexity of EDHOC should not distract from the security objectives such a protocol must satsify. EDHOC is an authentication protocol that is designed to be deployed on billions of IoT devices, and any security vulnerability in the design of the protocol would be difficult to fix and give adversaries a powerful platform to launch distributed attacks. Thus, we conduct a rigorous and mechanized security analysis of EDHOC, develop a formal model of the respective symmetric and asymmetric modes of EDHOC, and verify them in the protocol verifier ProVerif [3]. Our model allows the former to feed into the latter, i.e. keys established during the asymmetric mode can be used as long-term keys in the symmetric mode. In our formalization, we are able to verify that the protocol preserves *authentication, identity protection, secrecy and integrity of encrypted application data, and perfect forward secrecy of established sessions.*

Related work. The formal analysis of security standards is an active area of research. Bhargavan et al. [6] present an analysis of the TLS 1.3 draft 18, deriving from a reference implementation both ProVerif and CryptoVerif [7] models; a comprehensive analysis of TLS 1.3 draft 21 has been presented by Cremers at al. [8] using the Tamarin prover [9]. These works have served as an inspiration when analyzing the EDHOC protocol. Also, Meadows presented a formal analysis of the Internet Key Exchange protocol [10] using the NRL protocol verifier, to which EDHOC is related.

Structure of this paper. The paper proceeds as follows: Section 2 presents the EDHOC protocol as of draft version 08; Section 3 presents our modeling of the EDHOC protocol in ProVerif and our findings are presented in Section 4. Finally, we conclude in Section 5.

2 The Protocol

EDHOC is an authenticated key exchange protocol of the SIGMA-I family. It is a three-message exchange between an *initiator (U)* and a *responder (V)* that establishes a Diffie-Hellman shared secret between the two parties. Being a SIGMA-I protocol, each party can check the other identity without revealing it to a passive attacker, and the initiator identity is also protected from an active attacker.

Krawczyk presented the rationale behind the choices of SIGMA in [5], and Canetti and Krawczyk analysed it formally in [11]; we refer the interested reader to those papers for an extended presentation of the general scheme.

Before we dive into presenting the details of EDHOC, we first show the SIGMA-I protocol using authenticated encryption with associated data (AEAD), which we denote with $aead^{a}_{k}\{m\}$, using key k, optional authenticated data a and encrypting message m. We use exponentials instead of elliptic curve notation, as this make the paper easier to read. SIGMA-I follows this three-message exchange:

$$U \rightarrow V : g^{x} \tag{$\Sigma 1$}$$

$$V \rightarrow U : g^{y}, aead_{K_2}\{ID_V, sign_V(g^{x}, g^{y})\} \tag{$\Sigma 2$}$$

$$U \rightarrow V : aead_{K_3}\{ID_U, sign_U(g^{y}, g^{x})\} \tag{$\Sigma 3$}$$

The initiator first generates a fresh ephemeral public key for the session g^{x} and sends it to the responder in message ($\Sigma 1$); analogously, the responder generates their own ephemeral public key g^{y}. From the Diffie-Hellman shared secret $(g^{x})^{y}$ they can then derive the two encryption keys K_2 and K_3.[1] K_2 is used to encrypt the public identity ID_V together with the signature by the responder V of the two ephemeral keys (g^{x}, g^{y}).

Upon receiving message ($\Sigma 2$), the initiator can also derive K_2 and K_3, check the identity of the responder, and from that produce message ($\Sigma 3$) containing the encrypted signature of (g^{y}, g^{x}) and their identity.

Intuitively, the two signatures of message ($\Sigma 2$) and ($\Sigma 3$) ensure that the two parties U and V agree on the ephemeral keys g^{x} and g^{y} if they are freshly generated at each run of the protocol. Alternatively, if the concrete protocol reuses ephemeral keys over sessions at the expense of forward secrecy to—like EDHOC for example—save computations, it must include two public nonces in the signature for the agreement to hold. Furthermore, authenticated encryption is critical to bind the knowledge of the Diffie-Hellman shared secret $(g^{x})^{y}$ to the identities of V and U, as both K_2 and K_3 are derived from it; it also protects the responder's identity from a passive attacker, and the initiator's from an active one[2].

2.1 EDHOC with Asymmetric Keys

Figure 1 shows the asymmetric mode of operation for EDHOC. It implements the SIGMA-I protocol with a few added details. On top of the ephemeral Diffie-Hellman half-keys—which we now denote as E_U and E_V—EDHOC adds the following parameters to each session:

[1] See also Figures 1 and 2.

[2] The original SIGMA-I protocol uses a message authentication code (MAC), and then encrypts the signature and the authentication code with a symmetric encryption scheme for identity protection and binding: the use of authenticated encryption here serves this combined purpose.

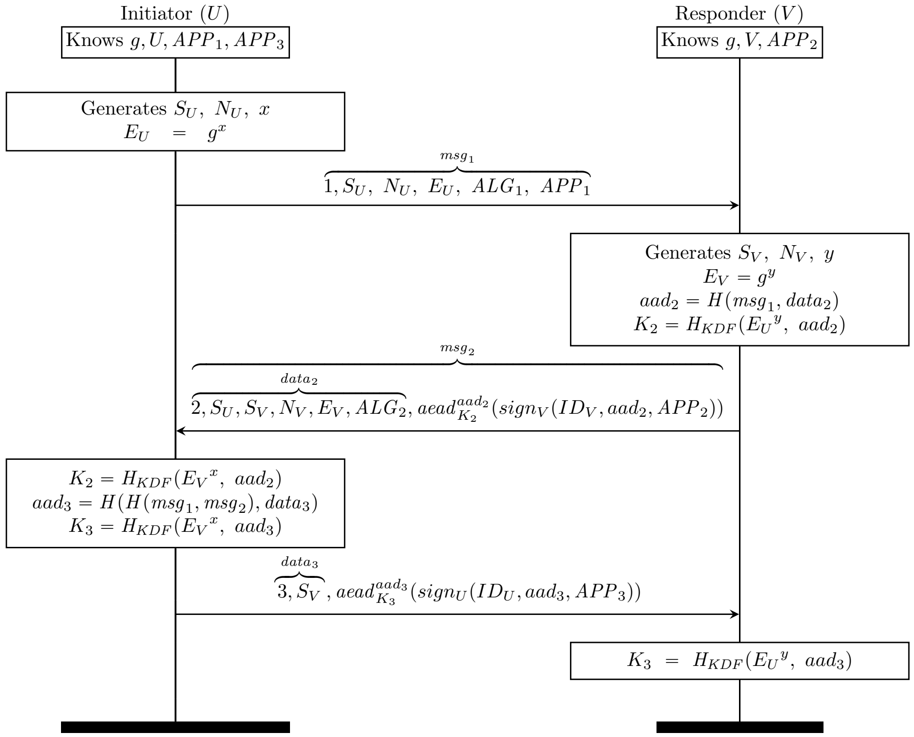
Initiator (U)
Knows g, U, APP_1, APP_3
Responder (V)
Knows g, V, APP_2
Generates $S_U,\ N_U,\ x$
$E_U = g^x$
$\overbrace{1, S_U,\ N_U,\ E_U,\ ALG_1,\ APP_1}^{msg_1}$
Generates $S_V,\ N_V,\ y$
$E_V = g^y$
$aad_2 = H(msg_1, data_2)$
$K_2 = H_{KDF}({E_U}^y,\ aad_2)$
$\overbrace{\overbrace{2, S_U, S_V, N_V, E_V, ALG_2}^{data_2}, aead_{K_2}^{aad_2}(sign_V(ID_V, aad_2, APP_2))}^{msg_2}$
$K_2 = H_{KDF}({E_V}^x,\ aad_2)$
$aad_3 = H(H(msg_1, msg_2), data_3)$
$K_3 = H_{KDF}({E_V}^x,\ aad_3)$
$\overbrace{3, S_V}^{data_3}, aead_{K_3}^{aad_3}(sign_U(ID_U, aad_3, APP_3))$
$K_3 = H_{KDF}({E_U}^y,\ aad_3)$

Fig. 1. Model of the asymmetric variant of EDHOC

- the constants 1, 2, and 3 in front of each message used for tagging;
- two session identifiers S_U, S_V that can be reused across sessions (e.g. to resume a previous session using a pre-shared key in the symmetric mode);
- two nonces N_U, N_V that must be fresh for each new session, relieving the requirement to generate fresh Diffie-Hellman half keys each time;
- the parameter ALG_1 that includes the names of all elliptic curves supported for the Diffie-Hellman key agreement, the supported Hashed Key Derivation Functions (HKDF) and authenticated encryption algorithms; similarly, the parameter ALG_2 includes the selected elliptic curve, HKDF and AEAD functions; we use H and $aead$ to denote these two functions, respectively;
- authentication data for each encrypted message (aad_2 and aad_3);
- optionally unencrypted application data APP_1 on the first message, and encrypted application data APP_2 and APP_3.

All messages in EDHOC are encoded using the CBOR Object Signing and Encryption standard (COSE) [12], which in turn uses the Compact Binary Object Representation [13], a binary alternative to the more common JSON web format. For the automatic security proof that we discuss in Section 3 we chose to abstract away from COSE and leave a mechanization to future work. The algorithm negotiation parameters aim to ensure compatibility with future versions by allowing flexibility in the choice of algorithms, however this being the first edition of the EDHOC standard, it mandates a fixed set of algorithms that are currently deemed secure: ECDH-SS, HKDF-256 [14], AES-CCM-64-64-128, and EdDSA from the COSE standard.

As the EDHOC key establishment protocol is intended to be combined with the OSCORE standard [4], it must derive a new OSCORE session key from the Diffie-Hellman session key, which by construction is ensured to be different from K_2 and K_3 by using the algorithm identifier "EDHOC OSCORE Master Secret" when applying the HKDF function.

Finally, the EDHOC asymmetric key exchange can produce a pre-shared key for further communication that is also ensured to be different from all other keys by using the algorithm identifier "EDHOC PSK Chaining" when applying the HKDF function.

2.2 EDHOC with Pre-shared Symmetric Keys

When two devices have a pre-shared key in EDHOC, either by establishing one through the PSK Chaining mode, or by being deployed with one, they can run the symmetric protocol, shown in Figure 2.

In the symmetric variant, the public identities of U and V are not used. Instead, the protocol relies on the presence of a pre-shared key PSK between U and V, identified by the value of KID. Overall, the symmetric variant runs similarly to the asymmetric one, except for the following:

- the keys K_i, $i \in \{2, 3\}$ are derived from the Diffie-Hellman shared secret $(g^x)^y$, the authentication data aad_i, and the pre-shared key PSK;
- there is no signature scheme involved to certify the identities of U and V, since their identities are already fixed by the identification of PSK.

Analogously to the asymmetric variant, the symmetric mode of EDHOC should guarantee secrecy of the established session key, authentication, identity protection, perfect forward secrecy, integrity protection of the application data APP_1, and secrecy of the application data APP_2 and APP_3.

However, since the pre-shared keys identify the two parties that share them, and are in turn identified by the KID parameter, even a passive attacker can link multiple sessions pertaining to the same two principals by observing KID. The standard at its current revision suggests that party U and V anonymize KID to avoid this attack, though it does not specify how the two parties should realize such anonymization. We will see next that the guarantees for the two variants of the protocol are in fact slightly different, not just regarding the claims of identity protection, but also regarding the claims of perfect forward secrecy, and integrity and secrecy protection of application data.

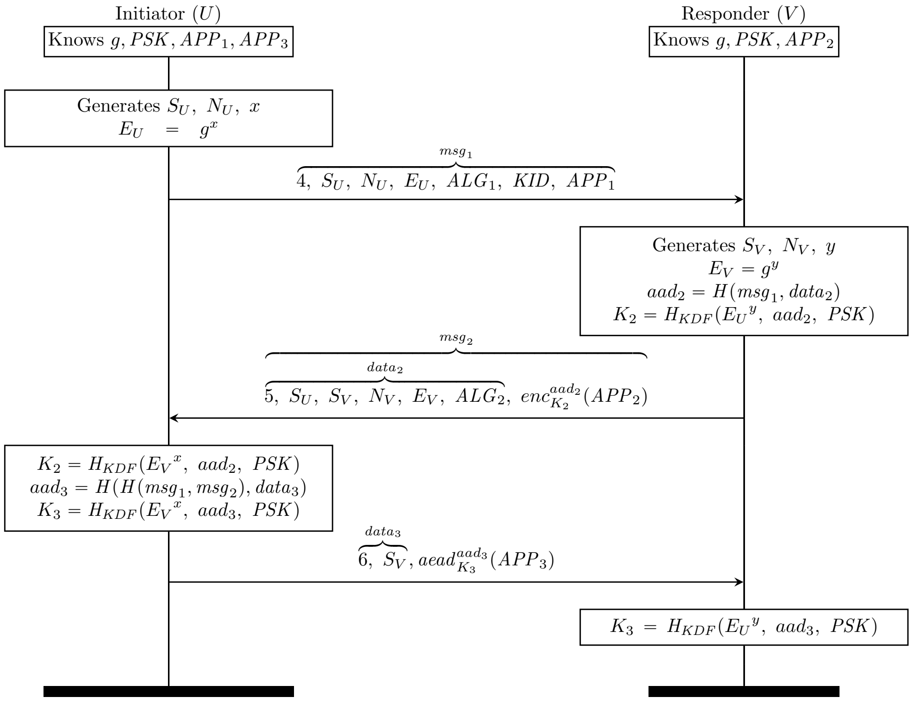
Initiator (U)
Knows g, PSK, APP_1, APP_3
Responder (V)
Knows g, PSK, APP_2
Generates $S_U,\ N_U,\ x$
$E_U = g^x$
$\overbrace{4,\ S_U,\ N_U,\ E_U,\ ALG_1,\ KID,\ APP_1}^{msg_1}$
Generates $S_V,\ N_V,\ y$
$E_V = g^y$
$aad_2 = H(msg_1, data_2)$
$K_2 = H_{KDF}({E_U}^y,\ aad_2,\ PSK)$
$\overbrace{\overbrace{5,\ S_U,\ S_V,\ N_V,\ E_V,\ ALG_2}^{data_2},\ enc_{K_2}^{aad_2}(APP_2)}^{msg_2}$
$K_2 = H_{KDF}({E_V}^x,\ aad_2,\ PSK)$
$aad_3 = H(H(msg_1, msg_2), data_3)$
$K_3 = H_{KDF}({E_V}^x,\ aad_3,\ PSK)$
$\overbrace{6,\ S_V}^{data_3}, aead_{K_3}^{aad_3}(APP_3)$
$K_3 = H_{KDF}({E_U}^y,\ aad_3,\ PSK)$

Fig. 2. Model of the symmetric variant of EDHOC

2.3 Properties

The current draft describes the security guarantees in Section 8 (Security Considerations) [1]. Here we summarize the security claims of the original text and integrate them with the results that we prove in our models, which we present Section 4.

Perfect forward secrecy. EDHOC, being part of the SIGMA-I family of protocols, should provide secrecy of the established session keys for past sessions, even when the long term keys are leaked. We check this for the session key derived for the OSCORE protocol. The same does not apply to K_2, as we discuss later.

Identity protection. Thanks to the use of authenticated encryption, EDHOC protects the initiator's identity from an active adversary, and the responder's identity from a passive adversary. However in symmetric variant reusing the same key identifier *KID* allows a passive attacker to correlate multiple sessions.[3]

[3] As discussed in Section 2.2, draft 08 warns against the reuse of *KID*, but does not prescribe a standard mechanism to avoid such reuse.

Protection of application data. Section 8 of the EDHOC draft 08 makes the following claims regarding the protection of application data:

> Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to APP_1 and APP_2 in the asymmetric case, and APP_1 and KID in the symmetric case.

And further down in the section:

> Party U and V must make sure that unprotected data does not trigger any harmful actions. In particular, this applies to APP_1 in the asymmetric case, and APP_1 and KID in the symmetric case. Party V should be aware that replays of EDHOC msg_1 cannot be detected unless previous nonces are stored.

Our claim is that these sentences do not reflect the actual guarantees that EDHOC offers. In fact, disregarding the issue on KID that we already discussed, all application data is protected in some way, though *the guarantees vary* between APP_1, APP_2 and APP_3, *according to how far the protocol has progressed*, according to *the variant* of the protocol that is being considered—being it *symmetric or asymmetric*—and finally also depending on whether we are checking for *integrity, secrecy or perfect forward secrecy.*

Table 1 summarizes the guarantees of EDHOC. The columns in this table identify the following guarantees: *Secrecy* denotes whether a specific piece of application data is guaranteed to be a secret between the two parties at the time it is sent; *Secrecy (at completion)* denotes whether the secrecy claim can be made if the protocol reaches completion; *PFS* denotes whether the protocol maintains secrecy of application data even in case that the long term keys are revealed; *PFS (at completion)* denotes whether the secrecy of a piece of application data is guaranteed in the case that the long term keys are revealed if the protocol reaches completion; finally *Integrity* denotes whether the other party can check the authenticity of a particular piece of application data, similarly *Integrity (at completion)* refers to whether the integrity of said application data can be trusted if the protocol completes successfully. For each case, the table shows a check mark if the particular piece of application data has the desired property, a cross if the property doesn't hold, and a dash where not relevant (i.e. secrecy of unencrypted data).

Note that the integrity of APP_1 is guaranteed if the protocol completes successfully. Furthermore, the standard does not clearly specify what "unprotected data" actually means, referring to APP_2 as unprotected data for the asymmetric variant, even though it is encrypted, while not for the symmetric case. As we will see, the guarantees that we get from APP_2 are rather interesting: one can rely on APP_2 being secret only at protocol completion in both the symmetric and the asymmetric case; when an active attacker interferes with the protocol run (and thus the protocol fails to reach completion), the attacker can obtain APP_2

Variant	Data	Secrecy	(at completion)	PFS	(at completion)	Integrity	(at completion)
Asymmetric	APP_1	–	–	–	–	✗	✓
	APP_2	✗	✓	✗	✓	✓	✓
	APP_3	✓	✓	✓	✓	✓	✓
Symmetric	APP_1	–	–	–	–	✗	✓
	APP_2	✓	✓	✗	✓	✓	✓
	APP_3	✓	✓	✓	✓	✓	✓

Table 1. Secrecy, PFS and integrity of application data.

for the asymmetric variant, but not for the symmetric one. However if the long term keys are leaked, there is an active attack for APP_2 also in the symmetric variant of the protocol, where the attacker interferes with the transmission of message 1 by injecting their own ephemeral key, and then learn the content of APP_2 when the long-term pre-shared key is leaked. This can however be avoided if the protocol reaches completion and hereby validating the authenticity of both principals.

Understanding the subtleties of these guarantees is a rather non-trivial task. In fact we advocate that the standard should claim that APP_2 is not confidential in both variants of the protocol, and to avoid potential implementation mistakes, the authors have decided to move APP_2 outside the encryption. On the other hand one can rely on the integrity of both APP_1 and APP_2 when the protocol reaches completion, which allows to relax the claims of EDHOC draft 08. That data can be used to perform irreversible actions, but only at the transmission of the third message.

3 Modeling EDHOC in Proverif

We model EDHOC in ProVerif [3], a symbolic protocol verifier supporting unbounded number of sessions and with support for Diffie-Hellman equational theories. ProVerif's logical foundation is the applied π-calculus [15], which we introduce and explain alongside our presentation of the EDHOC models. The four processes of Figure 4 and Figure 5 represent the Initiator and Responder roles of the protocol, in the symmetric and asymmetric variants, respectively. We consider a unified model where the asymmetric and the symmetric variants run in parallel, with unbounded numbers of principals and instances of each principal, that interact in any combination with each other. Furthermore, all long term private keys and pre-shared keys are revealed after the protocol has completed.

The equational theory and adversary model. By working on the symbolic Dolev-Yao model [16], we abstract away from concrete cryptographic primitives and their representation. Instead, we assume that the cryptographic primitives are "perfect", following common practice when working with ProVerif [17].

The Dolev-Yao model puts the attacker in control of the communication channel: it can intercept all communication between the honest principals, drop and inject messages, construct and decompose them using the known cryptographic primitives, and generate new secrets (keys, nonces etc.). It is a powerful adversary, though its capabilities end there: The Dolev-Yao attacker cannot decrypt messages without the proper keys, invert hash functions, etc. For the purpose of an efficient analysis, several defining features attributed to real cryptographic constructions are simply ignored, for example the extensionality of hash functions or the malleability of encryption schemes.

Let us thus define the symbols that model our cryptographic primitives, along with their arities:

$$\begin{aligned}\{&\mathit{aeadEncrypt}/3, \mathit{aeadDecrypt}/3, \mathit{decrypt}/2, \mathit{pk}/1, \mathit{sign}/2, \mathit{verify}/2\\ &\mathit{id}/1, \mathit{HKDF}/4, \mathit{empty}/0, \mathit{getAlgorithm}/1, \mathit{hash}/1, g/0, \mathit{exp}/2\}\end{aligned}$$

and the equations that hold between them:

$$\begin{aligned}\{&\mathit{aeadDecrypt}(\mathit{aeadEncrypt}(x, y, \mathit{aad}), y, \mathit{aad}) = x,\\ &\mathit{decrypt}(\mathit{aeadEncrypt}(x, y, \mathit{aad}), y) = x,\\ &\mathit{verify}(\mathit{sign}(x, y), \mathit{pk}(y)) = x,\\ &\mathit{getAlgorithm}(\mathit{HKDF}(x, \mathit{aad}, y, \mathit{algID})) = \mathit{algID},\\ &\mathit{exp}(\mathit{exp}(g, x), y) = \mathit{exp}(\mathit{exp}(g, y), x)\}\end{aligned}$$

The first equation implements the symmetric authenticated encryption scheme with associated data (*aead* in the protocol presentation): when a message is decrypted with key y the authentication data aad is also verified. The second equation models decryption without verifying the authenticity of the message. The third equation models a public signature scheme, where the signature of a message x with the private key y can be checked with the corresponding public key. The function H_{KDF} derives a key from the Diffie-Hellman shared secret, some authentication data, an optional symmetric key (we use the symbol *empty* to denote a missing key), and the algorithm—or purpose—for which it will be used. The function *getAlgorithm* is used to retrieve the algorithm identifier, and distinguish between keys with different purposes. This is a modelling artefact to aid termination of the tool, avoiding loops that re-insert pre-shared keys between parties that already ran the symmetric protocol (see lines 42-44 of the listing in Figure 5). No equation is defined for the hash function *hash*, since it is irreversible. Finally, the last equation models the commutative property of Diffie-Hellman groups.

The equations that we introduced so far are useful for the sake of automated verification: all but the last one are subterm convergent, and hence can be interpreted from left to right leading to a decidable equational theory; the last one can also be handled as a special case by ProVerif [18].

The main process is defined in Figure 3. It puts in parallel multiple replicated sub-processes ($P \mid Q$ represents the two processes P and Q running in parallel, and the bang operator ! represents unbounded copies of the same process).

The first process produces the identities of the unbounded principals: a fresh host U and their secret key skU are created in lines 2-3, using the **new** construction. The public key is constructed through the **let** binding and assigned

```
 1 process
 2   (!new U: host;
 3     new skU: skey;
 4     let pkU = pk(skU) in
 5     out(s, (U, skU, pkU)); out(s, (U, skU, pkU));
 6     out(c, (pkU)); phase 1; out(c, skU) ) |
 7   (!in(s, (U:host, skU:skey, pkU:pkey));
 8     in(s, (V:host, skV:skey, pkV:pkey));
 9     new random: bitstring;
10     let PSK = HKDF(g, random, empty, EDHOC_PRESHARED) in
11     out(s2, (U, V, PSK)); out(s2, (U, V, PSK)); phase 1; out(c, PSK) ) |
12   (!in(s, (U:host, skU:skey, pkU:pkey));
13     in(s, (V:host, skV:skey, pkV:pkey));
14     (initiatorAsym(U, V, skU, pkU, pkV) |
15      responderAsym(V, U, skV, pkV, pkU)) ) |
16   (!in(s2, (U:host, V:host, PSK:key));
17     (initiatorSym(U, V, PSK) |
18      responderSym(V, U, PSK)) )
```

Fig. 3. Main ProVerif process

to *pkU* in line 4. Finally, lines 5 and 6 output on the secret channel s the tuple (U, skU, pkU), then output the public key *pkU* on the public channel c, and through the use of the **phase** construct reveal the secret key *skU* after the protocol ended, that will later be used for checking Perfect Forward Secrecy.[4]

The replicated process in lines 7-11 models the out-of-bounds registration of pre-shared keys for the symmetric variant of the protocol: it inputs the principals U and V from the secret channel s, generates a fresh random value *random* and creates the pre-shared key *PSK* in line 10, which is output in line 11 on another secret channel $s2$ that maintains the mapping between the two principals and their shared key. Also *PSK* is output on the public channel in phase 1.

Finally, lines 12 through 18 of Figure 3 call the asymmetric and symmetric variants of the protocol—shown in Figure 1 and Figure 2—after binding the relevant data through the secret channels s and $s2$. We will not describe in detail those processes, as they closely follow the presentation of Section 2 and we hope the reader is by now acquainted with the language. We will however present their salient features.

3.1 Asymmetric Variant

Starting with the asymmetric variant of EDHOC in Figure 4, the initiator process is parameterized by their own identity U and that of the responder V it is

[4] The use of **phase** is a ProVerif-specific extension to the Applied Pi-calculus, which intuitively disables a process in a later phase to interact with processes from previous phases, though the attacker's information is carried through phases. For a more detailed description of how phases work, we refer to the ProVerif manual [3].

```
let initiatorAsym(U: host, V: host, skU: skey, pkU: pkey, pkV: pkey) =
  new x: exponent; let E_U = exp(g, x) in new S_U: bitstring;
  new N_U: nonce; new APP_1A: bitstring; new APP_3A: bitstring;
  event startInitiatorA(U, V, E_U, APP_1A, APP_3A);
  let msg_1: bitstring = (T1, S_U, N_U, E_U, APP_1A) in
  out(c, msg_1); in(c, msg_2: bitstring);
  let (data_2: bitstring, COSE_ENC_2: bitstring) = msg_2 in
  let (=T2, =S_U, xS_V: bitstring, N_V: nonce, xE_V: G) = data_2 in
  let aad_2: bitstring = hash((msg_1, data_2)) in
  let K: G = exp(xE_V, x) in
  let K_2: key = HKDF(K, aad_2, empty, EDHOC) in
  let signature_2: bitstring = aeadDecrypt(COSE_ENC_2, K_2, aad_2) in
  let (=idR(pkV), =aad_2, APP_2A: bitstring) = verify(signature_2, pkV) in
  event midInitiatorA(U, V, xE_V, APP_2A);
  let data_3: bitstring = (T3, xS_V) in
  let aad_3: bitstring = hash((hash((msg_1, msg_2)), data_3)) in
  let signature_3: bitstring = sign((idI(pk(skU)), aad_3, APP_3A), skU) in
  let K_3: key = HKDF(K, aad_3, empty, EDHOC) in
  let COSE_ENC_3: bitstring = aeadEncrypt(signature_3, K_3, aad_3) in
  let msg_3: bitstring = (data_3, COSE_ENC_3) in
  out(c, msg_3);
  event endInitiatorA(U, V, xE_V, APP_2A).

let responderAsym(V: host, U: host, skV: skey, pkV: pkey, pkU: pkey) =
  new y: exponent; let E_V: G = exp(g, y) in new S_V: bitstring;
  new N_V: nonce; new APP_2A: bitstring; new APP_2A': bitstring;
  event startResponderA(U, V, E_V, APP_2A);
  in(c, msg_1: bitstring);
  let (=T1, xS_U: bitstring, xN_U: nonce, xE_U: G, APP_1A: bitstring) = msg_1 in
  let data_2: bitstring = (T2, xS_U, S_V, N_V, E_V) in
  let aad_2: bitstring = hash((msg_1, data_2)) in
  let signature_2: bitstring = sign((idR(pk(skV)), aad_2, APP_2A), skV) in
  let K: G = exp(xE_U, y) in
  let K_2: key = HKDF(K, aad_2, empty, EDHOC) in
  let COSE_ENC_2: bitstring = aeadEncrypt(signature_2, K_2, aad_2) in
  let msg_2: bitstring = (data_2, COSE_ENC_2) in
  out(c, msg_2); in(c, msg_3: bitstring);
  let (data_3: bitstring, COSE_ENC_3: bitstring) = msg_3 in
  let (=T3, =S_V) = data_3 in
  let aad_3: bitstring = hash((hash((msg_1, msg_2)), data_3)) in
  let K_3: key = HKDF(K, aad_3, empty, EDHOC) in
  let signature_3: bitstring = aeadDecrypt(COSE_ENC_3, K_3, aad_3) in
  let (=idI(pkU), =aad_3, APP_3A: bitstring) = verify(signature_3, pkU) in
  event endResponderA(U, V, xE_U, APP_1A, APP_3A);
  let signature_2': bitstring = sign((idR(pkV), aad_2, APP_2A'), skV) in
  let COSE_ENC_2': bitstring = aeadEncrypt(signature_2', K_2, aad_2) in
  let msg_2': bitstring = (data_2, COSE_ENC_2') in
  out(c2, msg_2');
  let PSK' = HKDF(K, hash(msg_3), empty, EDHOC_PSK_Chaining) in
  out(s2, (U, V, PSK')).
```

Fig. 4. Asymmetric protocol

supposed to talk to, along with their private *skU* and corresponding public key *pkU*, as well as the responder public key *pkV*. A dual set of parameters is provided to the responder process to put them in communication.

The use of *id*(*pkU*) and *id*(*pkV*) is perhaps noteworthy: these are constructed using the private constructor *id*, and then used to check whether the attacker has learned the identity of one of the principals in the protocol. Even though the attacker has access to all public keys, they cannot construct these terms themselves, hence they need to learn them from the protocol. This allows to check whether the attacker can learn the identity of the each principal by checking if the attacker can obtain the information created with either of the two functions: *id*(*pkU*) or *id*(*pkV*).

In order to check agreement properties, the initiator and responder processes are annotated with the events *startInitiatorA*, *midInitiatorA*, *endInitiatorA* and *startResponderA*, *endResponderA*, respectively. The lack of a corresponding *midResponderA* is due to the first message being unprotected by any cryptographic mechanism, therefore the agreement at that point will trivially not hold.

In order to check whether the secrecy and PFS hold after the completion of the protocol, a duplicate of message two is created in the lines 45 through 48 of Figure 4. This message will be sent on a different channel that the processes do not listen to but the attacker does. It is hereby possible to check if the attacker can obtain the application data of message 2 after the completion of the protocol.

Finally, at the end of the responder process on line 49 of Figure 4, a pre-shared key PSK' is generated from the Diffie-Hellman shared secret, and then inserted into channel $s2$, which serves as a key-store for U and V.

3.2 Symmetric Variant

Figure 4 shows the symmetric variant of the protocol. It is parameterized by the identities of the hosts and the pre-shared key PSK that is used to establish the session. Like for the asymmetric variant, it is annotated with events that mark various steps of the protocol, with a lack of a *mid*-event for the responder. Also the events for the privacy of the initiator and responder are missing in this case, since the symmetric variant does not use at all public keys.

To check the secrecy and PFS properties after the completion of the protocol, the symmetric version similar to the asymmetric version creates a duplicate of the second message in the lines 39–41 of Figure 4.

Similarly to the asymmetric version, the responder process is also responsible for inserting a new pre-shared key PSK' derived from the Diffie-Hellman secret in lines 42–44, making use of the “EDHOC PSK Chaining” mode of the standard. Note the conditional in line 42: in our model we do not allow inserting a key derived using “EDHOC PSK Chaining” from one that was itself derive using the same technique. Doing so leads to the tool not terminating.

```
let initiatorSym(U: host, V: host, PSK: key) =
  new x: exponent; let E_U: G = exp(g, x) in new S_U: bitstring;
  new N_U: nonce; new APP_1S: bitstring; new APP_3S: bitstring;
  event startInitiatorS(U, V, E_U, APP_1S, APP_3S);
  let msg_1: bitstring = (T4, S_U, N_U, E_U, APP_1S) in
  out(c, msg_1); in(c, msg_2: bitstring);
  let (data_2: bitstring, COSE_ENC_2: bitstring) = msg_2 in
  let (=T5, =S_U, xS_V: bitstring, xN_V: nonce, xE_V: G) = data_2 in
  let aad_2: bitstring = hash((msg_1, data_2)) in
  let K: G = exp(xE_V, x) in
  let K_2: key = HKDF(K, aad_2, PSK, EDHOC) in
  let (APP_2S: bitstring) = aeadDecrypt(COSE_ENC_2, K_2, aad_2) in
  event midInitiatorS(U, V, xE_V, APP_2S);
  let data_3: bitstring = (T6, xS_V) in
  let aad_3: bitstring = hash((hash((msg_1, msg_2)), data_3)) in
  let K_3: key = HKDF(K, aad_3, PSK, EDHOC) in
  let msg_3: bitstring = (data_3, aeadEncrypt(APP_3S, K_3, aad_3)) in
  out(c, msg_3);
  event endInitiatorS(U, V, xE_V, APP_2S).

let responderSym(V: host, U: host, PSK: key) =
  new y: exponent; let E_V: G = exp(g, y) in new S_V: bitstring;
  new N_V: nonce; new APP_2S: bitstring; new APP_2S': bitstring;
  event startResponderS(U, V, E_V, APP_2S);
  in(c, msg_1: bitstring);
  let (=T4, xS_U: bitstring, N_U: nonce, xE_U: G, APP_1S: bitstring) = msg_1 in
  let data_2: bitstring = (T5, xS_U, S_V, N_V, E_V) in
  let aad_2: bitstring = hash((msg_1, data_2)) in
  let K: G = exp(xE_U, y) in
  let K_2: key = HKDF(K, aad_2, PSK, EDHOC) in
  let msg_2: bitstring = (data_2, aeadEncrypt(APP_2S, K_2, aad_2)) in
  out(c, msg_2); in(c, msg_3: bitstring);
  let (data_3: bitstring, COSE_ENC_3: bitstring) = msg_3 in
  let (=T6, =S_V) = data_3 in
  let aad_3: bitstring = hash((hash((msg_1, msg_2)), data_3)) in
  let K_3: key = HKDF(K, aad_3, PSK, EDHOC) in
  let (APP_3S: bitstring) = aeadDecrypt(COSE_ENC_3, K_3, aad_3) in
  event endResponderS(U, V, xE_U, APP_1S, APP_3S);
  let COSE_ENC_2': bitstring = aeadEncrypt(APP_2S', K_2, aad_2) in
  let msg_2': bitstring = (data_2, COSE_ENC_2') in
  out(c2, msg_2');
  if(getAlgorithm(PSK) = EDHOC_PRESHARED) then
  let PSK' = HKDF(K, hash(msg_3), PSK, EDHOC_PSK_Chaining) in
  out(s2, (U, V, PSK')).
```

Fig. 5. Symmetric protocol

4 Security Properties

In this Section, we present the definitions used to check the properties that we claim in Section 2.3, in terms of ProVerif queries.

Identity protection against an active adversary The protocol reveals the identity of the initiator against an active adversary if the attacker can obtain the term $idI(pkU)$ for any public key pkU registered to an honest party. Likewise, the protocol reveals the identity of the responder from an active adversary if the attacker can obtain the term $idR(pkU)$ for any public key pkU registered to an honest party. As we expect, ProVerif confirms that the identity of the initiator kept private, whereas the identity of the responder is revealed.

Secrecy of data, perfect forward secrecy We measure secrecy and perfect forward secrecy of APP_2 and APP_3. That is, we check whether the attacker is able to obtain the application data from the second and the third message of the protocol, in both the symmetric and asymmetric variants. In the asymmetric variant, we find that APP_2 is not secret unless the session completes, while APP_3 is secret. These properties are maintained for completed sessions even after revealing the long-term keys. In the symmetric variant, APP_2 is secret (but not after revealing the long-term keys with incomplete sessions), and APP_3 is secret even after revealing long-term keys. These results match our claims on Table 1 for the columns “Secrecy” and “PFS” as well as their corresponding “(at completion)” columns.

Strong authentication In our model, all correspondence agreements between the initiator and the responder hold. Note that in both versions, when the initiator U ends the protocol, or even when they receive the second message, we have no assurance that the responder intended to talk to U, thus we leave freedom of choice for another U' on these correspondence checks. This is obviously acceptable: because of the unprotected nature of the first message the responder has no way to check the identity of the initiator until it reaches the end of the protocol. This is reflected by the other side of the correspondence, that matches in all parameters, including the application data. Hence we can confirm the integrity of all application data at the completion of the protocol, and also the integrity of APP_2 when the initiator receives it, in both cases. However since the responder is not certain of the identity of the initiator when sending APP_2, that payload should not contain any information about that is confidential for them.

5 Conclusion

The EDHOC protocol is the result of a clean slate design of an authentication protocol for IoT devices that is light weight, easy to implement, and that does not have any legacy modes. The protocol is currently under consideration for standardization, with the next step being to prove the standardization body that the protocol is secure. In this work, we describe a formal analysis of the protocol in the symbolic model using the ProVerif tool. We have identified security issues and hidden assumptions, which have led to further refinements of the EDHOC protocol. ProVerif has been an excellent tool to conduct this work, although, our model had to be carefully engineered, not to get in trouble with the well-known over-approximations that ProVerif models entail. In general we would recommend that any protocol that is used in security sensitive domains and considered for standardization should be modelled using formal methods and verified using mechanized reasoning tools, such as ProVerif.

Future work The protocol has been analyzed alone, whereas it is intended to be used in conjunction with other protocols, including the secure data transport standard "Object Security for Constrained RESTful Environments" (OSCORE [4]), for which EDHOC creates an explicit context (i.e. session keys). Protocol composition is not automatically guaranteed, and is an active area of research in the theory of protocol security. Even though it is reassuring to know that the OSCORE session key is never used nor is derivable from an EDHOC key (though it is derivable from the Diffie-Hellman exchange), it would be interesting to conduct a formal analysis of those standards that are meant to be used in conjunction. Other involved standards that would require further analysis would be the CBOR binary format [13], and the COSE [12] standard of encryption formats, which are used by EDHOC for formatting messages, and mimick the JSON web standard and JOSE encryption standard that sits on top of it. Furthermore, since the protocol is at its first version and only mandates the use of algorithms that are currently deemed secure, we have not modelled downgrade attacks. It will be helpful to extend this model to consider downgrade attacks, once new versions of the EDHOC are released.

The next iteration of the EDHOC protocol, draft 09 [19], has incorporated our findings, and the authors decided to move away from encrypting the application data contained in the second message, in an effort to simplify the protocol and offer weaker—but clearer—security guarantees. This work has provided useful input to the designers of EDHOC, and the authors intend to evolve the models with the evolution of the standard, so as to provide useful input and higher assurance on the correctness of its design.

References

1. Göran Selander, John Mattsson, F.P.: Ephemeral Diffie-Hellman over COSE (EDHOC). https://tools.ietf.org/html/draft-selander-ace-cose-ecdhe-08 (2018) [Online; accessed 10 May. 2018].
2. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-28, Internet Engineering Task Force (2018) Work in Progress.
3. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial (2018)

4. Selander, G., Mattsson, J., Palombini, F., Seitz, L.: Object Security for Constrained RESTful Environments (OSCORE). Internet-Draft draft-ietf-core-object-security-13, Internet Engineering Task Force (2018) Work in Progress.
5. Jim Schaad, A.C.: CBOR Object Signing and Encryption (COSE). https://tools.ietf.org/html/rfc8152 (2010) [Online; accessed 10 May. 2018].
6. Krawczyk, H.: SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Annual International Cryptology Conference, Springer (2003) 400–425
7. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: Security and Privacy (SP), 2017 IEEE Symposium on, IEEE (2017) 483–502
8. Blanchet, B.: CryptoVerif: Computationally sound mechanized prover for cryptographic protocols. In: Dagstuhl seminar "Formal Protocol Verification Applied. Volume 117. (2007)
9. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A comprehensive symbolic analysis of TLS 1.3. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM (2017) 1773–1788
10. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: International Conference on Computer Aided Verification, Springer (2013) 696–701
11. Meadows, C.: Analysis of the internet key exchange protocol using the NRL protocol analyzer. In: Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on, IEEE (1999) 216–231
12. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Annual International Cryptology Conference, Springer (2002) 143–161
13. Bormann, C.: Concise Binary Object Representation (CBOR). https://tools.ietf.org/html/rfc7049 (2013) [Online; accessed 10 May. 2018].
14. Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (2010)
15. Abadi, M., Blanchet, B., Fournet, C.: The applied pi calculus: Mobile values, new names, and secure communication. J. ACM **65** (2018) 1:1–1:41
16. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29** (1983) 198–208
17. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Foundations and Trends in Privacy and Security **1** (2016) 1–135
18. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Computer Security Foundations Symposium (CSF), 2012 IEEE 25th, IEEE (2012) 78–94
19. Selander, G., Mattsson, J., Palombini, F.: Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-selander-ace-cose-ecdhe-09, Internet Engineering Task Force (2018) Work in Progress.