LAKE
Internet-Draft
Intended status: Informational
Expires: 2 September 2026

C. Papon
C. Onete
XLIM UMR CNRS 7252 - Limoges University

1 March 2026

Post-Quantum EDHOC - Initiator and Responder using signature and/or KEM
draft-papon-lake-pq-edhoc-00

Abstract

This document specifies two extensions to the Ephemeral Diffie-Hellman over COSE (EDHOC). These two protocol versions aim to provide quantum-resistance to the original EDHOC protocol, while reducing message-complexity with respect to parallel drafts. The document defines: (1) a 3-message quantum-resistant EDHOC proposal when the Initiator knows the Responder; in this version, the Initiator authenticates using a signature, while the Responder uses a KEM; (2) a 3-or-4-message quantum-resistant EDHOC proposal, which proposes a tradeoff between message-complexity and computational overhead.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 September 2026.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology and Requirements Language 3
1.2. Definitions 4
1.2.1. KEMs (Key Encapsulation Mechanisms) 4
1.2.2. Digital Signature 4
2. Post-Quantum EDHOC when the Initiator knows the Responder (PQ-EDHOC-IKR) 5
2.1. Motivation 5
2.2. PQ-EDHOC-IKR protocol overview 5
2.2.1. PQ-EDHOC-IKR protocol description 5
2.2.2. Key Derivation Schedule 9
2.2.3. Additional explanations 11
2.3. Analysis 12
3. KEM & Sign Authentication for Post-Quantum EDHOC 13
3.1. Motivation 13
3.2. First case: Initiator signs, Responder KEM and signs . . 14
3.2.1. Protocol overview 14
3.2.2. Protocol description 15
3.2.3. Associated key derivation schedule 18
3.2.4. Additional explanations 20
3.2.5. Analysis 21
3.3. Second case: Initiator KEM and signs, Responder signs . . 21
3.3.1. Protocol overview 21
3.3.2. Protocol description 22
3.3.3. Associated key derivation schedule 25
3.3.4. Additional explanations 27
3.3.5. Analysis 28
3.4. Third case: Initiator and Responder KEM and sign - version 1 29
3.4.1. Protocol overview 29
3.4.2. Protocol description 29
3.4.3. Associated key derivation schedule 32
3.4.4. Additional explanations 34
3.4.5. Analysis 35
3.5. Third case: Initiator and Responder KEM and sign - version 2 36
3.5.1. Protocol overview 36
3.5.2. Protocol description 36
3.5.3. Associated key derivation schedule 40
3.5.4. Additional explanations 41
3.5.5. Analysis 42

4. Security Considerations 43
4.1. Forward Secrecy 44
4.2. Identity protection 45
4.3. Downgrade Protection 46
4.4. Transcript Hash Binding 46
4.5. External Authorization Data (EAD) 46
5. IANA Considerations 46
6. References 46
6.1. Normative References 46
6.2. Informative References 48
Authors' Addresses 48

1. Introduction

This document aims to present new alternatives for rendering the Ephemeral Diffie-Hellman Over COSE (EDHOC) protocol quantum-resistant. The main goal is to reduce the number of messages exchanged during the handshake, using a combination of KEMs and signatures, as well as Post-Quantum Cryptography (PQC) cipher suites, for secure key-exchange and authentication.

In this draft we:

* propose, building on [I-D.pocero-authkem-ikr-edhoc], a version of quantum-resistant EDHOC where the Initiator knows the Responder, but in a scenario where the Initiator authenticates with a signature (ML-DSA) and the Responder with a KEM (ML-KEM).

* propose, building on [I-D.pocero-authkem-edhoc], a version of the protocol that requires, depending on the case, either only 3 or 4 mandatory messages. This second version reduces the message overhead, but comes at an additional computational overhead for (at least) one of the two parties.

The two proposals in this draft can be viewed to extend (in the case of the first bullet-point) and respectively to provide an alternative (in the case of the second bullet-point) to the current proposal for quantum-resistant EDHOC.

1.1. Terminology and Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in EDHOC [RFC9528], CBOR [RFC8949], CBOR Sequences [RFC8742], COSE Structures and Processing [RFC9052] and COSE Algorithms [RFC9053].

1.2. Definitions

1.2.1. KEMs (Key Encapsulation Mechanisms)

A Key Encapsulation Mechanism (KEM) consists of three algorithms:

* KEM.KeyGen: a probabilistic key generation algorithm, which takes as input a security parameter k and outputs a key pair (kem.sk, kem.pk), where kem.sk is secret and kem.pk is public.

* KEM.Encapsulation: a probabilistic encapsulation algorithm, which takes as input a KEM public key kem.pk, and generates a pair (ss, kem.ct), where ss is the shared-secret and kem.ct is a ciphertext.

* KEM.Decapsulation: a deterministic decapsulation algorithm, which takes as input a KEM secret key kem.sk and a KEM ciphertext kem.ct, and outputs the shared-secret ss (associated to kem.ct).

In practice, if Alice and Bob use a KEM, Alice generates the public and private keys, sends the public key to Bob, who generates the shared-secret and ciphertext. He then returns the latter to Alice, who can subsequently recover the same shared-secret using her private key and the ciphertext.

1.2.2. Digital Signature

A Digital Signature (DS) scheme consists of three algorithms:

* DS.KeyGen: a probabilistic key generation algorithm, which takes as input a security parameter k and outputs a key pair (sign.sk, sign.pk), where sign.sk is secret and sign.pk is public.

* DS.Sign: a probabilistic algorithm, which takes as input a signing secret key sign.sk and a message m, and generates a signature s.

* DS.Verify: a deterministic algorithm, which takes as input a signing public key sign.pk, a message m and a signature s, and outputs 1 if the signature is valid for the message, and 0 otherwise.

In practice, if Alice and Bob use a DS scheme, Alice generates the public and private keys, and sends the public key to Bob. Later, Alice sends a message together with a signature to Bob. Bob can then verify the validity of the signature on the message under the public key.

2. Post-Quantum EDHOC when the Initiator knows the Responder (PQ-EDHOC-IKR)

2.1. Motivation

In [I-D.pocero-authkem-ikr-edhoc], the authors adopt an EDHOC approach for use cases where the Initiator already knows the identity of the Responder, in a post-quantum version using ephemeral and static KEMs for authentication. The Initiator, knowing the long-term public key of the Responder, can derive the shared-secret ss_R and compute a key, then encrypt part of the first message. It can thus send its identity directly and securely authenticate himself, since only the Responder with its long-term secret key can decrypt the first message. Knowing the Initiator’s identity, the Responder can derive the second shared-secret ss_I and continue the key derivation schedule. Sending the encrypted second message allows it to securely confirm its identity to the Initiator.

The key motivation behind our current proposal is a desire to reduce the message-complexity of quantum-resistant EDHOC. We propose an extension to draft proposal [I-D.pocero-authkem-ikr-edhoc], which allows the Initiator and Responder to authenticate using different mechanisms -- much as in the original EDHOC. In particular, while the Responder still authenticates using a KEM, in our proposal the Initiator will use a signature.

This approach comes with the following benefit: there is no need to partially encrypt the first message. Instead, during the third message, the Initiator calculates a MAC and signs it. Upon receiving the encrypted third message, the Responder, after decryption, authenticates the Initiator in the usual EDHOC manner.

2.2. PQ-EDHOC-IKR protocol overview

2.2.1. PQ-EDHOC-IKR protocol description

We present here a high-level description of our PQ-EDHOC-IKR where the Initiator authenticates with a signature and the Responder with a KEM.

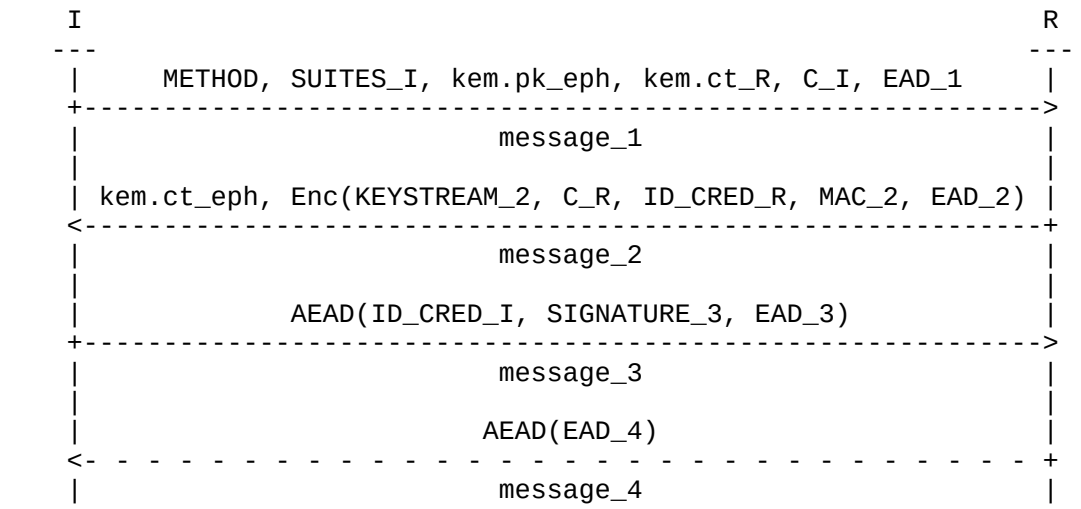


Figure 1: PQ-EDHOC-IKR (I sign, R kem) message flow.

2.2.1.1. Formatting and sending message_1

As in the usual EDHOC protocol, the first message (message_1) consists of:

* METHOD --> as specified in [RFC9528] it is an integer specifying the authentication method the Initiator wants to use;

* SUITES_I --> it consists of an ordered set of algorithms supported by the Initiator and formatted as specified in [RFC9528];

* C_I (and also as C_R, which will appears later) --> the Connection Identifiers chosen by the Initiator (C_I) and by the Responder (C_R) as specified in [RFC9528];

* EAD_1 (and also EAD_2, EAD_3 and EAD_4, which will appear later) -> External Authorization Data, respectively included in message_1, message_2, message_3 (and optionally) message_4, and formatted as specified in [RFC9528];

* kem.pk_eph --> the Ephemeral KEM public key generated by the Initiator;

* kem.ct_R --> based on the Responder long-term KEM public key, the Initiator computes ss_R and kem.ct_R with the KEM.Encapsulation algorithm. He keeps secret ss_R to compute, later, PRK_3e2m, and sends kem.ct_R to the Responder.

2.2.1.2. Processing message_1, formatting and sending message_2

On the reception of the first message, the Responder first recovers ss_R thanks to his long-term KEM secret key kem.sk_R and kem.ct_R, using the KEM.Decapsulation algorithm (if the decapsulation process fails, he aborts). He then proceeds as in the original EDHOC protocol with elements METHOD, SUITES_I, C_I and EAD_1. Finally, using kem.pk_eph and the KEM.Encapsulation algorithm, he computes the ephemeral ciphertext kem.ct_eph and the ephemeral shared-secret ss_eph.

The Responder selects its Connection Identifier C_R as specified in [RFC9528]. He then computes:

* TH_2 = H(kem.ct_eph, H(message_1));

* PRK_2e = EDHOC_Extract(TH_2, ss_eph).

And also, as in [RFC9528]:

* KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);

* SALT_3e2m = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length).

Compared to [RFC9528], the computation of PRK_3e2m is modified as follows :

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R).

The Responder now computes MAC_2 and assembles PLAINTEXT_2:

* MAC_2 = EDHOC_KDF(PRK_3e2m, 2, C_R, ID_CRED_R, TH_2, EAD_2, mac_length_2);

* PLAINTEXT_2 = (C_R, ID_CRED_R, MAC_2, EAD_2).

So the second message consists of:

* kem.ct_eph --> the ephemeral ciphertext obtained with kem.pk_eph;

* CIPHERTEXT_2 = PLAINTEXT_2 XOR KEYSTREAM_2.

2.2.1.3. Processing message_2, formatting and sending message_3

On reception of the second message, the Initiator, using kemp.ct_eph, can compute the ephemeral shared-secret ss_eph. As the Responder did, he computes TH_2, PRK_2e and KEYSTREAM_2. He can now decipher and retrieve : PLAINTEXT_2 = CIPHERTEXT_2 XOR KEYSTREAM_2.

Thanks to ID_CRED_R, the Initiator verifies the validity of the long-term KEM public key of the Responder, kemp.pk_R, and computes SALT_3e2m and PRK_3e2m, using the shared-secret ss_R he generates at the beginning of the exchange.

At this point the Initiator is able to authenticate the Responder (at least, make sure he is talking to the endpoint he hopes to talk to). For that, he computes, as the Responder did, the MAC MAC_2. If it matches with the one he received, then he properly authenticated the Responder. Otherwise he aborts.

It is now up to the Initiator to authenticate himself. To do so, the Initiator computes the following elements:

* TH_3 = H(TH_2, PLAINTEXT_2, ID_CRED_R);

* K_3 = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);

* IV_3 = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);

* MAC_3 = EDHOC_KDF(PRK_3e2m, 6, ID_CRED_I, TH_3, EAD_3, mac_length_3).

In this version of the protocol, the Initiator authenticates itself to the Responder with a signature:

* SIGNATURE_3 = DS.Sign(sign.sk_I, (ID_CRED_I, TH_3, EAD_3, MAC_3, sign_length))

where sign.sk_I is the long-term signing private key of the Initiator.

Setting PLAINTEXT_3 = (ID_CRED_I, SIGNATURE_3, EAD_3), the Initiator ciphers PLAINTEXT_3 with the AEAD encryption algorithm negociated in SUITES_I.

2.2.1.4. Processing message_3

On reception of message_3, the Responder computes TH_3, K_3, and IV_3 as the Initiator did, and deciphers CIPHERTEXT_3 with the AEAD decryption algorithm. With PLAINTEXT_3, he can compute MAC_3 on his side, and verify the signature:

* DS.Verify(sign.pk_I, (ID_CRED_I, TH_3, EAD_3, MAC_3, sign_length), SIGNATURE_3)

where sign.pk_I is the long-term signing public key of the Initiator. If the verification algorithm returns 1, the Responder properly authenticated the Initiator. Otherwise he aborts.

2.2.1.5. Optionally formatting, sending and receiving message_4

If the Responder decides of a fourth mandatory message, he then computes the following elements:

* TH_4 = H(TH_3, PLAINTEXT_3, ID_CRED_I);

* K_4 = EDHOC_KDF(PRK_3e2m, 8, TH_4, key_length);

* IV_4 = EDHOC_KDF(PRK_3e2m, 9, TH_4, iv_length).

Using the AEAD encryption algorithm, he ciphers PLAINTEXT_4 = EAD_4 and sends it to the Initiator. The latter computes TH_4, K_4, IV_4, and deciphers CIPHERTEXT_4 thanks to the AEAD decryption algorithm.

2.2.1.6. Computing the session key PRK_out

It doesn’t matter if there is a fourth mandatory message, in any case, both the Initiator and the Responder, in order to compute the key PRK_out, have to calculate the fourth transcript hash:

* TH_4 = H(TH_3, PLAINTEXT_3, ID_CRED_I).

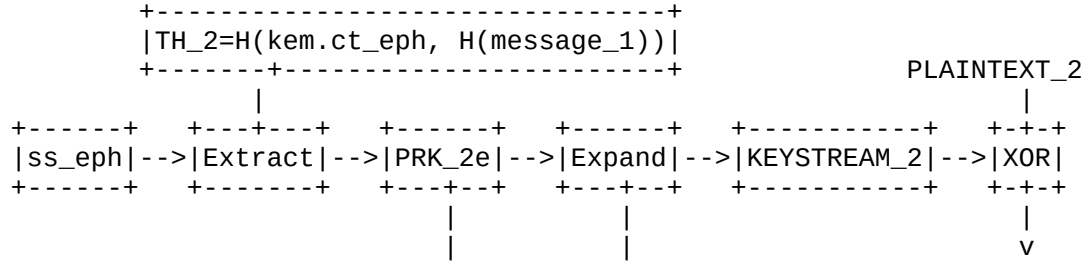
With that element, using the EDHOC_KDF algorithm, they both obtain:

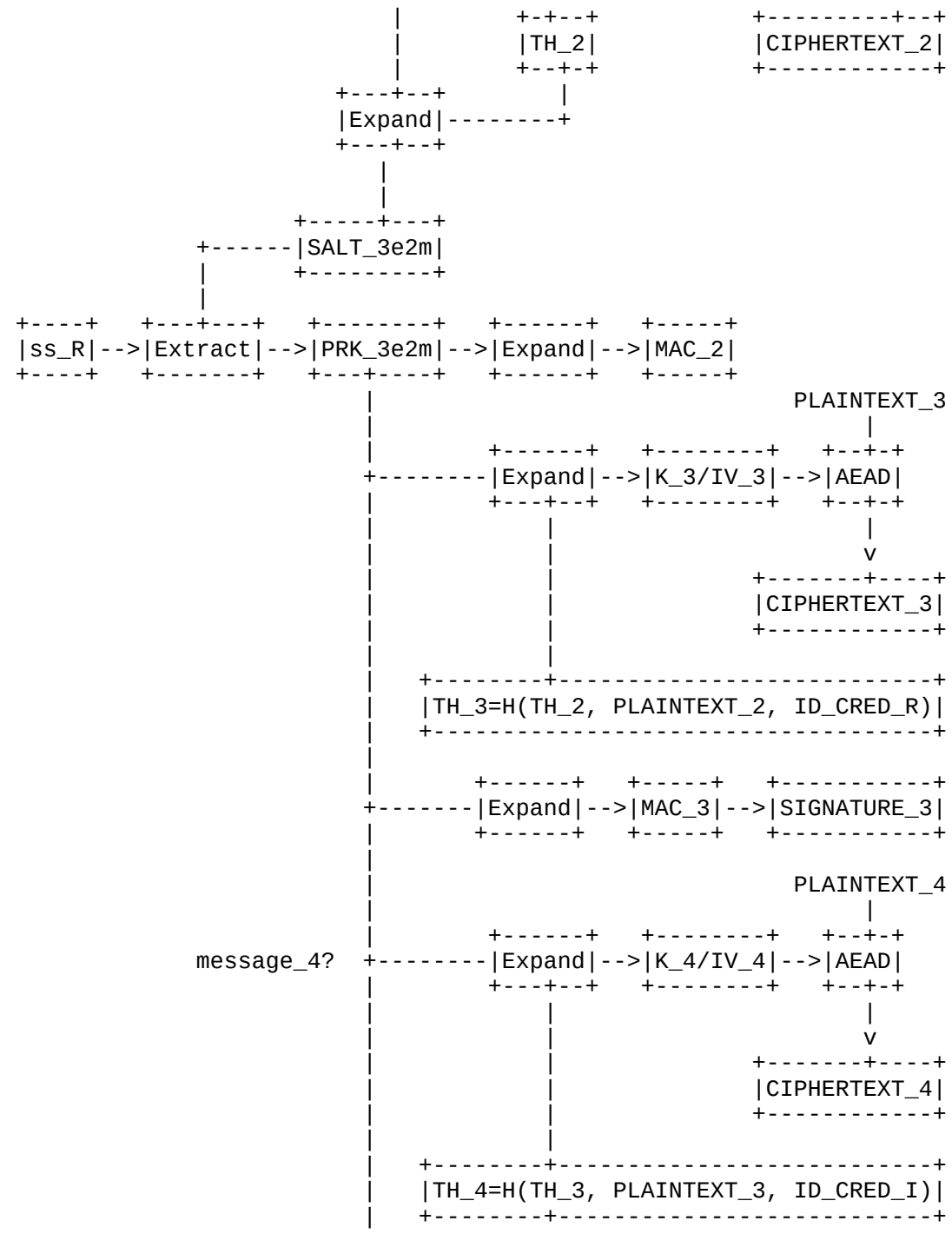
* PRK_out = EDHOC_KDF(PRK_3e2m, 7, TH_4, hash_length)

which is the desired session key, and the authentication process is fully achieved.

2.2.2. Key Derivation Schedule

In this section we summarize the key derivation operations that appears throughout the protocol.




|TH_2|
|CIPHERTEXT_2|
|Expand|
|SALT_3e2m|
|ss_R|
|Extract|
|PRK_3e2m|
|Expand|
|MAC_2|
PLAINTEXT_3
|Expand|
|K_3/IV_3|
|AEAD|
|CIPHERTEXT_3|
|TH_3=H(TH_2, PLAINTEXT_2, ID_CRED_R)|
|Expand|
|MAC_3|
|SIGNATURE_3|
PLAINTEXT_4
message_4?
|Expand|
|K_4/IV_4|
|AEAD|
|CIPHERTEXT_4|
|TH_4=H(TH_3, PLAINTEXT_3, ID_CRED_I)|

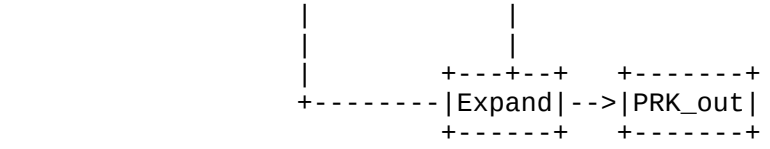


Figure 2: PQ-EDHOC-IKR (I sign, R kem) key derivation schedule.

So we can summarize the different computations as in [RFC9528]:

```
PRK_2e      = EDHOC_Extract(TH_2, ss_eph);
KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);
SALT_3e2m   = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);
PRK_3e2m    = EDHOC_Extract(SALT_3e2m, ss_R);
MAC_2       = EDHOC_KDF(PRK_3e2m, 2, context_2, mac_length_2)
              (with context_2 = C_R, ID_CRED_R, TH_2, EAD_2);
K_3         = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);
IV_3        = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);
MAC_3       = EDHOC_KDF(PRK_3e2m, 6, context_3, mac_length_3)
                    (with context_3 = ID_CRED_I, TH_3, EAD_3);
K_4         = EDHOC_KDF(PRK_3e2m, 8, TH_4, key_length);
IV_4        = EDHOC_KDF(PRK_3e2m, 9, TH_4, iv_length);
PRK_out     = EDHOC_KDF(PRK_3e2m, 7, TH_4, hash_length).
```

2.2.3. Additional explanations

We detail here the main elements that differ from the original EDHOC protocol.

2.2.3.1. Ephemeral KEM

As explained in [I-D.pocero-authkem-ikr-edhoc], the usual ephemeral Diffie-Hellman elements (x,g^x) and (y,g^y), are here replaced by an ephemeral KEM:

* At first, the Initiator generates a ephemeral KEM key pair (kem.sk_eph, kem.pk_eph) using the KEM.KeyGen(k) algorithm (where k is a security parameter);

* On reception of the ephemeral KEM public key kemp.pk_eph, the Responder generates a pair (ss_eph, kem.ct_eph) with the KEM.Encapsulation algorithm (with input kem.pk_eph). The element ss_eph is the ephemeral shared-secret, later used to derive a key, and kem.ct_eph is the ephemeral KEM ciphertext, used by the Initiator to retrieve the ephemeral shared-secret;

* On reception of the ephemeral KEM ciphertext, the Initiator recovers the ephemeral shared-secret ss_eph thanks to the KEM.Decapsulation’ algorithm (with inputkem.sk_ephandkem.ct_eph‘).

2.2.3.2. Authentication keys

For the Initiator, the authentication key MUST be a static signing key pair (sign.sk_I, sign.pk_I) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

For the Responder, the authentication key MUST be a static KEM key pair (kem.sk_R, kem.pk_R) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter).

2.2.3.3. Key derivation

In this version of the protocol, the EDHOC_Extract fonction is used to derive two keys, in a way that slightly differ from the original EDHOC protocol.

* PRK_2e = EDHOC_Extract(TH_2, ss_eph) --> the salt SHALL be TH_2 and the IKM SHALL be the ephemeral shared-secret ss_eph;

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R) --> the salt SHALL be SALT_3e2m directly derived from PRK_2e, and the IKM SHALL be the ’static’ shared-secret ss_R.

Concerning PRK_out, there is no modifications compared to the original EDHOC protocol.

2.3. Analysis

In summary, we retain here the use case proposed by [I-D.pocero-authkem-ikr-edhoc], where the Initiator already knows the Responder identity. Our protocol proposes a hybridization between the protocol proposed by [I-D.pocero-authkem-ikr-edhoc] and EDHOC method 1. In other words, Diffie-Hellman elements are replaced with KEMs, and the Initiator authenticates here using a signature (instead of a KEM). An important difference with the proposal of [I-D.pocero-authkem-ikr-edhoc] concerns the first message message_1, which in our version is simplified and remains closer to the standard EDHOC structure. Indeed, the Initiator authenticates via a signature at the third message. So he no longer needs to derive a key and encrypt part of the first message to securely send its identity. The second message also finds itself simplified. Since the Initiator does not use KEM in this situation, the Responder has no need to generate a shared-secret ss_I and send the KEM ciphertext kem.ct_I. The second message is therefore closer to the usual EDHOC structure.

Finally, the third message is almost ’identical’ to the third message of the EDHOC method 1 protocol. It seems important to note that complete and secure authentication is ensured here entirely in three messages, and the fourth message is optional. Regarding the Key Derivation Schedule, the changes are minor compared to [RFC9528] (adaptation of notations, especially for the IKM to take into account the post-quantum nature of the cryptographic material) and [I-D.pocero-authkem-ikr-edhoc] (simply one less key derivation, since the Initiator uses a signature rather than a static KEM). We will discuss security considerations further in this document. We will also soon present a byte comparison for each message.

3. KEM & Sign Authentication for Post-Quantum EDHOC

3.1. Motivation

Our second idea is to propose a tradeoff, allowing for a reduced number of messages in the quantum-resistant EDHOC handshake, which come at a slightly higher computational overhead compared to [I-D.pocero-authkem-edhoc].

Starting from the standard EDHOC protocol, method 0 allows for mutual authentication via signature between both users. As proposed in [I-D.pocero-authkem-edhoc], replacing classical signatures with post-quantum resistant signatures such as ML-DSA and ephemeral Diffie-Hellman elements with a KEM like ML-KEM seems reasonable to make EDHOC post-quantum resistant (even if this still needs to be proved). These two changes do not affect the number of mandatory messages, since the fundamental structure of the protocol is preserved (however a post-quantum signature will likely be more computationally expensive, just like a KEM). Things get complicated when trying to apply these modifications to the other three methods. Continuing on this track, we replace ephemeral Diffie-Hellman elements with an ephemeral KEM. This does not affect the usual structure of EDHOC. However, if we want to take it further and also replace long-term Diffie-Hellman elements with a KEM, managing asymmetric keys for the latter poses a problem when trying to preserve the authentication structure of EDHOC, namely a MAC derived from a long-term secret. As proposed in [I-D.pocero-authkem-edhoc], we go from 3 to 4 or 5 mandatory messages to achieve complete mutual authentication. Here, we want to explore a different path. Our solution aims to strike a balance between the number of mandatory messages and the computations performed by each endpoint.

The principle is as follows. Suppose that the Initiator authenticates with a signature and the Responder uses a KEM. In this case, we will need an obligatory fourth message (for the Initiator to authenticate the Responder via the MAC, as proposed in

[I-D.pocero-authkem-edhoc]). To avoid this obligatory fourth message and return to three, we propose removing the MAC MAC_2 used to authenticate the Responder (eliminating calculations on both sides) and replacing it with a signature from the Responder (adding calculations on both sides). To do this, the Responder signs an element (which should not be chosen randomly for security reasons and which must allow the Initiator to identify the session) and sends it directly (encrypted) in the second message. The Initiator can then immediately identify the Responder: it retrieves its identity from the encrypted message and verifies the signature. The rest of the protocol remains 'unchanged', and the calculation of PRK_out is possible directly after sending/receiving the third message (thus no more obligatory fourth message). The costs of calculations are compensated, but the delicate point remains the size of the second message (we will provide a bytes analysis for this message later). We decline this procedure when the Initiator wants to authenticate with a KEM and the Responder with a signature, or when both parties want to authenticate with KEMs. In these two latter cases, the number of mandatory messages compared to [I-D.pocero-authkem-edhoc] decreases from 5 to 4.

3.2. First case: Initiator signs, Responder KEM and signs

We describe below the equivalent of EDHOC method 1. The Initiator will authenticate with a signature, and the Responder with a KEM and a signature.

3.2.1. Protocol overview

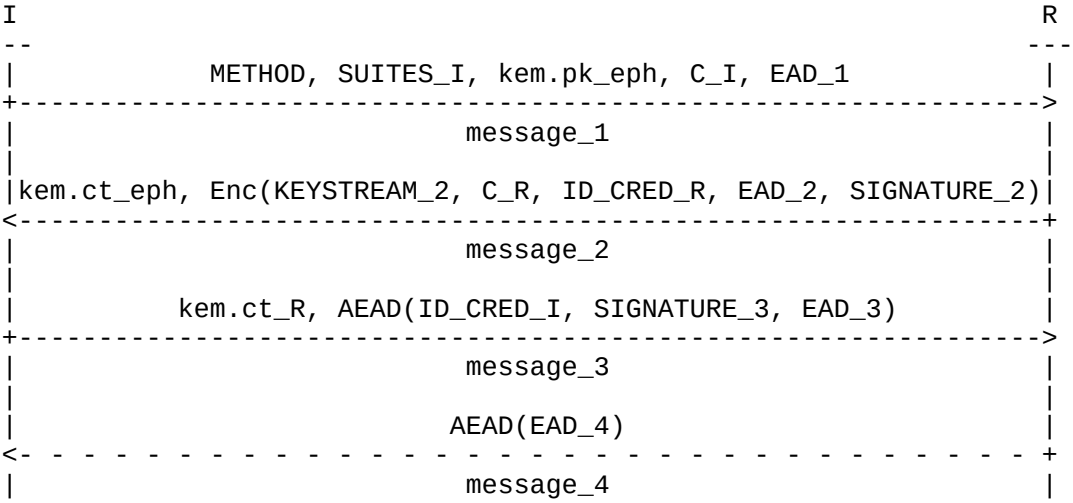


Figure 3: PQ-EDHOC, I Signs - R KEM & Signs message flow.

3.2.2. Protocol description

3.2.2.1. Formatting and sending message_1

As in the usual EDHOC protocol, the first message (message_1) consists of:

* METHOD --> as specified in [RFC9528] it is an integer specifying the authentication method the Initiator wants to use;

* SUITES_I --> it consists of an ordered set of algorithms supported by the Initiator and formatted as specified in [RFC9528];

* C_I (and also as C_R, which will appears later) --> the Connection Identifiers chosen by the Initiator (C_I) and by the Responder (C_R) as specified in [RFC9528];

* EAD_1 (and also EAD_2, EAD_3 and EAD_4, which will appear later) -> External Authorization Data, respectively included in message_1, message_2, message_3 (and optionally) message_4, and formatted as specified in [RFC9528];

* kem.pk_eph --> the Ephemeral KEM public key generated by the Initiator.

3.2.2.2. Processing message_1, formatting and sending message_2

On the reception of the first message, the Responder proceeds as in the original EDHOC protocol with elements METHOD, SUITES_I, C_I and EAD_1. In a second step, using kem.pk_eph and the KEM.Encapsulation algorithm, he computes the ephemeral ciphertext kem.ct_eph and the ephemeral shared-secret ss_eph.

The Responder selects its Connection Identifier C_R as specified in [RFC9528]. He then computes:

* TH_2 = H(kem.ct_eph, H(message_1));

* PRK_2e = EDHOC_Extract(TH_2, ss_eph);

* KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length).

The Responder now assembles PLAINTEXT_2:

* PLAINTEXT_2 = (C_R, ID_CRED_R, TH_2, EAD_2)

then signs this message :

* SIGNATURE_2 = DS.Sign(sign.sk_R, (PLAINTEXT_2, sign_length))

where sign.sk_R is the long-term signing private key of the Responder, and finally ciphers PLAINTEXT_2A = (PLAINTEXT_2, SIGNATURE_2):

* CIPHERTEXT_2 = PLAINTEXT_2A XOR KEYSTREAM_2.

So the second message then consists of:

* kem.ct_eph --> the ephemeral ciphertext obtained with kem.pk_eph;

* CIPHERTEXT_2.

Important note: let us mention that the element signed by the Responder, for security considerations during the security analysis, could be subject to slight changes. However, it serves here to illustrate the principle proposed here.

3.2.2.3. Processing message_2, formatting and sending message_3

On reception of the second message, the Initiator, using kemp.ct_eph, can compute the ephemeral shared-secret ss_eph. As the Responder did, he computes TH_2, PRK_2e and KEYSTREAM_2. He can now decipher and retrieve : PLAINTEXT_2A = CIPHERTEXT_2 XOR KEYSTREAM_2.

Thanks to ID_CRED_R, the Initiator obtains the long-term public keys of the Responder, kemp.pk_R and sign.pk_R. At first, he checks the validity of the signature of the Responder:

* DS.Verify(sign.pk_R, (PLAINTEXT_2, sign_length), SIGNATURE_2).

If the verification algorithm returns 1, the Initiator properly authenticated the Responder. Otherwise he aborts.

Assuming everything goes well, using the KEM encapsulation algorithm KEM.Encapsulation, and the long-term input material kem.pk_R of the Responder, the Initiator generated the couple (ss_R, kem.ct_R).

The shared-secret ss_R will then serve as IKM for the computation of PRK_3e2m:

* SALT_3e2m = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R).

It is now the turn of the Initiator to authenticate himself. To do so, he computes the following elements:

* TH_3 = H(TH_2, PLAINTEXT_2A, ID_CRED_R);

* K_3 = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);

* IV_3 = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);

* MAC_3 = EDHOC_KDF(PRK_3e2m, 6, ID_CRED_I, TH_3, EAD_3, mac_length_3)

as in the original EDHOC protocol. Then comes the signature:

* SIGNATURE_3 = DS.Sign(sign.sk_I, (ID_CRED_I, TH_3, EAD_3, MAC_3, sign_length))

where sign.sk_I is the long-term signing private key of the Initiator.

Setting PLAINTEXT_3 = (ID_CRED_I, SIGNATURE_3, EAD_3), the Initiator ciphers PLAINTEXT_3 with the AEAD encryption algorithm negociated in SUITES_I. The third message is then composed of:

* CIPHERTEXT_3;

* kem.ct_R.

3.2.2.4. Processing message_3

On reception of message_3, the Responder computes TH_3. He also used the KEM.Decapsulation algorithm with its long-term KEM private key kem.sk_R and the KEM ciphertext kem.ct_R, to obtain the shared-secret ss_R.

He can now compute SALT_3e2m, PRK_3e2m, K_3, and IV_3 as the Initiator did, and deciphers CIPHERTEXT_3 with the AEAD decryption algorithm. With PLAINTEXT_3, he calculates MAC_3 on his side, and verifies the signature:

* DS.Verify(sign.pk_I, (ID_CRED_I, TH_3, EAD_3, MAC_3, sign_length), SIGNATURE_3)

where sign.pk_I is the long-term signing public key of the Initiator. If the verification algorithm returns 1, the Initiator is properly authenticated to the Initiator. Otherwise the Responder aborts.

3.2.2.5. Optionally formatting, sending and receiving message_4

If the Responder decides of a fourth mandatory message, he then computes the following elements:

* TH_4 = H(TH_3, PLAINTEXT_3, ID_CRED_I);

* K_4 = EDHOC_KDF(PRK_3e2m, 8, TH_4, key_length);

* IV_4 = EDHOC_KDF(PRK_3e2m, 9, TH_4, iv_length).

Using the AEAD encryption algorithm, he ciphers PLAINTEXT_4 = EAD_4 and sends it to the Initiator. The latter computes TH_4, K_4, IV_4, and deciphers CIPHERTEXT_4 thanks to the AEAD decryption algorithm.

3.2.2.6. Computing the session key PRK_out

Here again, it doesn’t matter if there is a fourth mandatory message, in any case, both the Initiator and the Responder, in order to compute the key PRK_out, have to calculate the fourth transcript hash as in the original EDHOC protocol:

* TH_4 = H(TH_3, PLAINTEXT_3, ID_CRED_I).

With this element, using the EDHOC_KDF, they both obtain:

* PRK_out = EDHOC_KDF(PRK_3e2m, 7, TH_4, hash_length)

which is the desired session key, and the authentication process is fully achieved.

3.2.3. Associated key derivation schedule

In this section we present the key derivation schedule of the previously described protocol version.

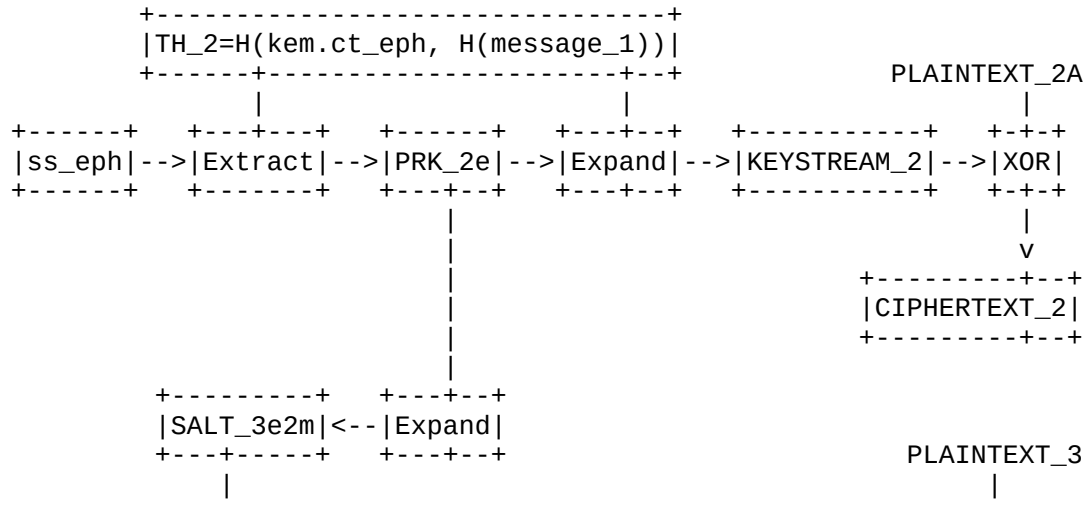


Figure 4: PQ-EDHOC, I Signs - R KEM & Signs key derivation schedule.

So we can summarize the different computations as previously:

```
PRK_2e      = EDHOC_Extract(TH_2, ss_eph);
KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);
SALT_3e2m   = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);
PRK_3e2m    = EDHOC_Extract(SALT_3e2m, ss_R);
K_3         = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);
IV_3        = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);
MAC_3       = EDHOC_KDF(PRK_3e2m, 6, context_3, mac_length_3)
                    (with context_3 = ID_CRED_I, TH_3, EAD_3);
K_4         = EDHOC_KDF(PRK_3e2m, 8, TH_4, key_length);
IV_4        = EDHOC_KDF(PRK_3e2m, 9, TH_4, iv_length);
PRK_out     = EDHOC_KDF(PRK_3e2m, 7, TH_4, hash_length).
```

3.2.4. Additional explanations

We detail here the main elements that differ from the original EDHOC protocol.

3.2.4.1. Ephemeral KEM

As previously explain the usual ephemeral Diffie-Hellman elements are replaced by an ephemeral KEM. In this protocol, it works exactly as we previously described.

3.2.4.2. Authentication keys

For the Initiator, the authentication key MUST be a static signing key pair (sign.sk_I, sign.pk_I) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

For the Responder, the authentication key MUST be: (1) a static KEM key pair (kem.sk_R, kem.pk_R) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter) *and* (2) a static signing key pair (sign.sk_R, sign.pk_R) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

3.2.4.3. Key derivation

In this version of the protocol, the keys PRK_2e and PRK_3e2m are obtain thanks to EDHOC_Extract fonction:

* PRK_2e = EDHOC_Extract(TH_2, ss_eph) --> the salt SHALL be TH_2 and the IKM SHALL be the ephemeral shared-secret ss_eph;

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R) --> the salt SHALL be SALT_3e2m directly derived from PRK_2e, and the IKM SHALL be the ’static’ shared-secret ss_R.

Concerning PRK_out, here again, there is no modifications compared to the original EDHOC protocol.

3.2.5. Analysis

In this version of the protocol, we assumed that the Initiator has a pair of static signature keys (sign.sk_I, sign.pk_I) and that the Responder has not only a pair of static signature keys (sign.sk_R, sign.pk_R) but also a pair of static KEM keys (kem.sk_R, kem.pk_R). This is not an unusual case. We can suppose that each user, when registering in a Public Key Infrastructure (PKI), provides pairs of keys of both types (signature and KEM), which they update regularly. The difference in our protocol is that the Responder is forced to use both types during the same AKE. From a technical point of view, the first message message_1 and the third message message_3 do not differ from those proposed in [I-D.pocero-authkem-edhoc]. Two major differences are worth noting regarding the messages. Firstly, here we have only an *optional* fourth message message_4. Secondly, our second message message_2 contains an additional signature SIGNATURE_2 in PLAINTEXT_2A (before ciphering) (we will propose a byte analysis of this message later). Finally, from a computational point of view, both the Initiator and the Responder no longer need to calculate the MAC MAC_2. In return, the Responder must sign a message, and the Initiator must verify this signature. Regarding the Key Derivation Schedule, aside from the IKM which adapts to post-quantum cryptographic material, we remain close to the one of EDHOC’s method 1. We thus have a tradeoff between the size of messages, calculation capabilities, and the number of messages.

3.3. Second case: Initiator KEM and signs, Responder signs

We now invert the roles and describe below the equivalent of EDHOC method 2. The Initiator will authenticate with a KEM and a signature, and the Responder only with a signature.

3.3.1. Protocol overview

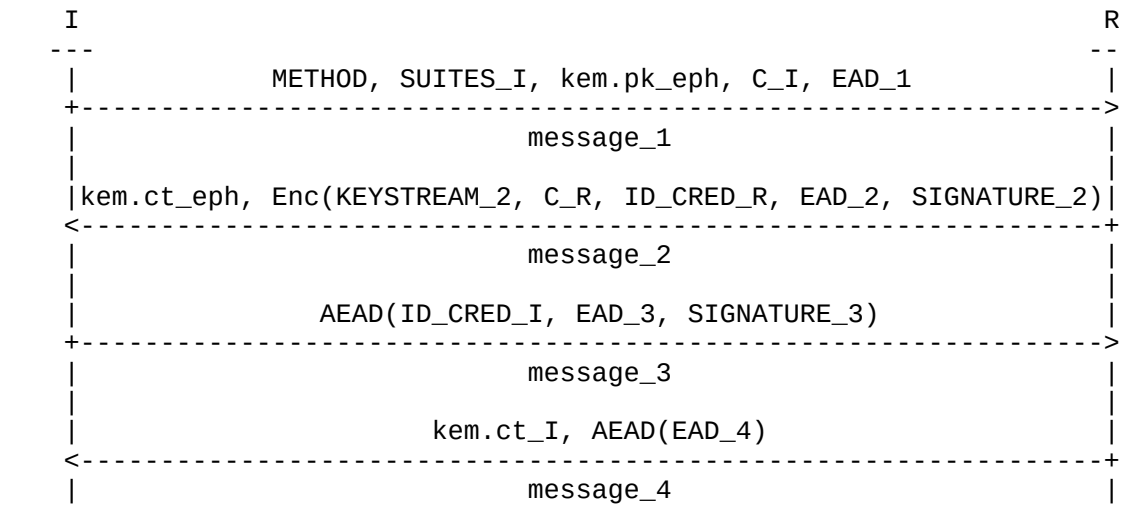


Figure 5: PQ-EDHOC, I KEM & Signs - R Signs message flow.

3.3.2. Protocol description

3.3.2.1. Formatting and sending message_1

As for our previous protocol version, the first message (message_1) consists of:

* METHOD --> as specified in [RFC9528] it is an integer specifying the authentication method the Initiator wants to use;

* SUITES_I --> it consists of an ordered set of algorithms supported by the Initiator and formatted as specified in [RFC9528];

* C_I (and also as C_R, which will appears later) --> the Connection Identifiers chosen by the Initiator (C_I) and by the Responder (C_R) as specified in [RFC9528];

* EAD_1 (and also EAD_2, EAD_3 and EAD_4, which will appear later) -> External Authorization Data, respectively included in message_1, message_2, message_3 (and optionally) message_4, and formatted as specified in [RFC9528];

* kem.pk_eph --> the Ephemeral KEM public key generated by the Initiator.

3.3.2.2. Processing message_1, formatting and sending message_2

On the reception of the first message, the Responder proceeds as in the original EDHOC protocol with elements METHOD, SUITES_I, C_I and EAD_1. In a second step, using kem.pk_eph and the KEM.Encapsulation algorithm, he computes the ephemeral ciphertext kem.ct_eph and the ephemeral shared-secret ss_eph.

The Responder selects its Connection Identifier C_R as specified in [RFC9528]. He then computes:

* TH_2 = H(kem.ct_eph, H(message_1));

* PRK_2e = EDHOC_Extract(TH_2, ss_eph);

* KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length).

In order to authenticate himself to the Initiator, in this version, similarly to the EDHOC method 2, the Responder computes a MAC MAC_2 and signs it with its long-term signing private key sign.sk_R:

* MAC_2 = EDHOC_KDF(PRK_2e, 2, C_R, ID_CRED_R, TH_2, EAD_2, mac_length_2);

* SIGNATURE_2 = DS.Sign(sign.sk_R, (C_R, ID_CRED_R, TH_2, EAD_2, MAC_2, sign_length)).

The Responder now assembles PLAINTEXT_2:

* PLAINTEXT_2 = (C_R, ID_CRED_R, SIGNATURE_2, EAD_2)

then ciphers this message :

* CIPHERTEXT_2 = PLAINTEXT_2 XOR KEYSTREAM_2

and finally sends message_2 composed of CIPHERTEXT_2 and kem.ct_eph. So, so far, except the ephemeral KEM material, everything works like in the usual EDHOC method 2 protocol.

3.3.2.3. Processing message_2, formatting and sending message_3

On reception of the second message, the Initiator, using kemp.ct_eph, can compute the ephemeral shared-secret ss_eph. As the Responder did, he computes TH_2, PRK_2e and KEYSTREAM_2. He can now decipher and retrieve : PLAINTEXT_2 = CIPHERTEXT_2 XOR KEYSTREAM_2.

Thanks to ID_CRED_R, the Initiator obtains the long-term signing public key of the Responder, sign.pk_R. He can then compute MAC_2 on its own, and check the validity of the signature of the Responder:

* DS.Verify(sign.pk_R, (C_R, ID_CRED_R, TH_2, EAD_2, MAC_2, sign_length), SIGNATURE_2).

If the verification algorithm returns 1, the Initiator properly authenticated the Responder. Otherwise he aborts.

Assuming everything goes well, the Initiator computes the following elements:

* TH_3 = H(TH_2, PLAINTEXT_2, ID_CRED_R);

* K_3 = EDHOC_KDF(PRK_2e, 3, TH_3, key_length);

* IV_3 = EDHOC_KDF(PRK_2e, 4, TH_3, iv_length).

Then, in order to authenticate, he forms PLAINTEXT_3 = (ID_CRED_I, TH_3, EAD_3) and signs it:

* SIGNATURE_3 = DS.Sign(sign.sk_I, (PLAINTEXT_3, sign_length)).

where sign.sk_I is the long-term signing private key of the Initiator. Finally he ciphers PLAINTEXT_3A = (PLAINTEXT_3, SIGNATURE_3) with the AEAD encryption algorithm negociated in SUITES_I, as CIPHERTEXT_3.

The third message is then composed of:

* CIPHERTEXT_3.

Important note: let us mention that the element signed by the Initiator, here again, for security considerations during the security analysis, could be subject to slight changes.

3.3.2.4. Processing message_3, formatting and sending message_4

On reception of message_3, the Responder computes TH_3, K_3 and IV_3, and deciphers CIPHERTEXT_3 with the AEAD decryption algorithm. With PLAINTEXT_3A, he obtains the Initiator’s long-term signing public key, and so he verifies the signature:

* DS.Verify(sign.pk_I, (PLAINTEXT_3, sign_length), SIGNATURE_3).

If the verification algorithm returns 1, the Initiator is properly authenticated to the Initiator. Otherwise the Responder aborts. With PLAINTEXT_3, the Responder also gets the Initiator's long-term KEM public key. He then uses the KEM.Encapsulation algorithm to generate ss_I and kem.ct_I.

Following that, he computes:

* TH_4 = H(TH_3, PLAINTEXT_3A, ID_CRED_I);

* SALT_4e3m = EDHOC_KDF(PRK_2e, 5, TH_4, hash_length);

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I);

* K_4 = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);

* IV_4 = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length).

Using the AEAD encryption algorithm, he ciphers PLAINTEXT_4 = EAD_4 and sends it to the Initiator, along with kem.ct_I. He then finishes the AKE by computing the final key PRK_out:

* PRK_out = EDHOC_KDF(PRK_4e3m, 7, TH_4, hash_length).

3.3.2.5. Processing message_4

In this situation the fourth message is mandatory. It is used to send the KEM ciphertext kem.ct_I to the Initiator (it cannot be shared before, since the Responder only learns the identity of the Initiator with the third message).

When receiving this last message, the Initiator computes the shared-secret ss_I thanks to the KEM.Decapsulation algorithm, and elements kem.sk_I, kem.ct_I. After that, he can compute TH_4, K_4, IV_4, and decipher CIPHERTEXT_4 thanks to the AEAD decryption algorithm. Once he had checked the well formedness of PLAINTEXT_4, he can finally compute PRK_out as the Responder did.

3.3.3. Associated key derivation schedule

In this section we present the key derivation schedule of the previously described protocol version.

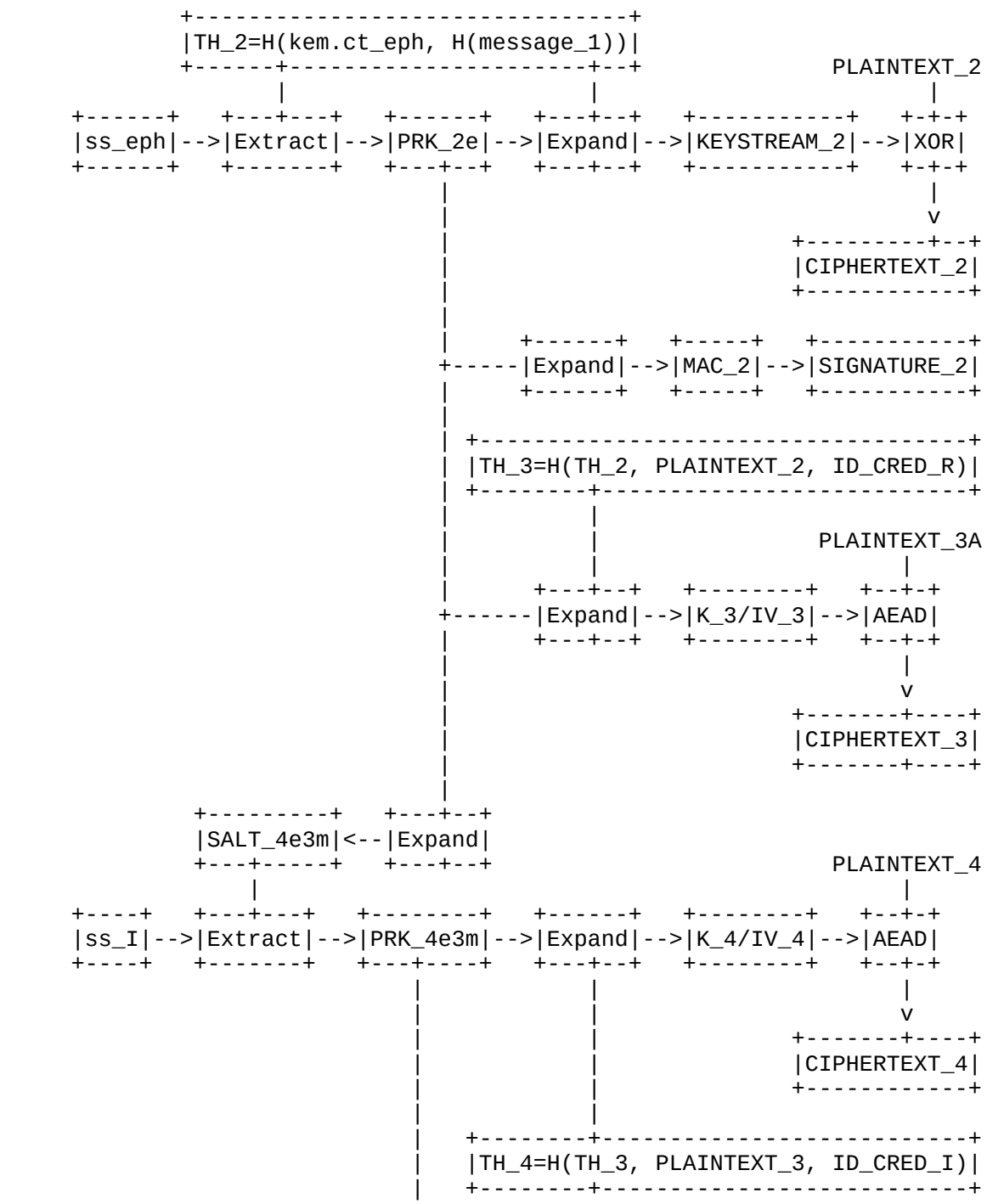

|TH_2=H(kem.ct_eph, H(message_1))|
PLAINTEXT_2
|ss_eph|
Extract
PRK_2e
Expand
KEYSTREAM_2
XOR
CIPHERTEXT_2
Expand
MAC_2
SIGNATURE_2
|TH_3=H(TH_2, PLAINTEXT_2, ID_CRED_R)|
PLAINTEXT_3A
Expand
K_3/IV_3
AEAD
CIPHERTEXT_3
SALT_4e3m
Expand
PLAINTEXT_4
|ss_I|
Extract
PRK_4e3m
Expand
K_4/IV_4
AEAD
CIPHERTEXT_4
|TH_4=H(TH_3, PLAINTEXT_3, ID_CRED_I)|

Figure 6: PQ-EDHOC, I KEM & Signs - R Signs key derivation schedule.

So we can summarize the different computations as previously:

```
PRK_2e      = EDHOC_Extract(TH_2, ss_eph);
KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);
MAC_2       = EDHOC_KDF(PRK_2e, 2, context_2, mac_length_2)
              (with context_2 = C_R, ID_CRED_R, TH_2, EAD_2);
K_3         = EDHOC_KDF(PRK_2e, 3, TH_3, key_length);
IV_3        = EDHOC_KDF(PRK_2e, 4, TH_3, iv_length);
SALT_4e3m   = EDHOC_KDF(PRK_2e, 5, TH_4, hash_length);
PRK_4e3m    = EDHOC_Extract(SALT_4e3m, ss_I);
K_4         = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);
IV_4        = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length);
PRK_out     = EDHOC_KDF(PRK_4e3m, 7, TH_4, hash_length).
```

3.3.4. Additional explanations

We detail here the main elements that differ from the original EDHOC protocol.

3.3.4.1. Ephemeral KEM

As previously explain the usual ephemeral Diffie-Hellman elements are replaced by an ephemeral KEM. In this protocol, it works exactly as we previously described.

3.3.4.2. Authentication keys

For the Initiator, the authentication key MUST be: (1) a static KEM key pair (kem.sk_I, kem.pk_I) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter) *and* (2) a static signing key pair (sign.sk_I, sign.pk_I) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

For the Responder, the authentication key MUST be a static signing key pair (sign.sk_R, sign.pk_R) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

3.3.4.3. Key derivation

In this version of the protocol, the keys PRK_2e and PRK_4e3m are obtain thanks to EDHOC_Extract fonction:

* PRK_2e = EDHOC_Extract(TH_2, ss_eph) --> the salt SHALL be TH_2 and the IKM SHALL be the ephemeral shared-secret ss_eph;

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I) --> the salt SHALL be SALT_4e3m directly derived from PRK_2e, and the IKM SHALL be the 'static' shared-secret ss_I.

Concerning PRK_out, here again, there is no modifications compared to the original EDHOC protocol. We can note that in this two previous cases, the Key Derivation Schedule remains close to the standard EDHOC Key Derivation Schedule of method 1 and 2.

3.3.5. Analysis

This is actually a mirror version of the previous protocol presented, in the sense that this time we assumed that the Responder has a pair of static signature keys (sign.sk_R, sign.pk_R) and that the Initiator has not only a pair of static signature keys (sign.sk_I, sign.pk_I) but also a pair of static KEM keys (kem.sk_I, kem.pk_I). For the same reasons, this is not an unusual case.

From a technical point of view, messages message_1,message_2 and message_4 do not differ from those proposed in [I-D.pocero-authkem-edhoc]. Here again, two major differences are worth noting regarding the messages. Firstly, here we no more have a fifth mandatory message message_5. Secondly, our third message message_3 contains an additional signature SIGNATURE_3 in PLAINTEXT_3A (before ciphering) (as this was the case for the previous protocol we propose, with SIGNATURE_2) (we will propose a byte analysis of this message later). Finally, from a computational point of view, both the Initiator and the Responder no longer need to calculate the MAC MAC_3. In return, in the same vein, the Initiator must sign a message, and the Responder must verify this signature, in order to authenticate the Initiator, and send him the KEM ciphertext kem.ct_I. Regarding the Key Derivation Schedule, aside from the IKM which adapts to post-quantum cryptographic material, we remain close to the one of EDHOC's method 2. So compared to [I-D.pocero-authkem-edhoc], we reduce the number of mandatory messages, from 5 to 4, and again we obtain a tradeoff between the size of messages, calculation capabilities, and the number of messages.

3.4. Third case: Initiator and Responder KEM and sign - version 1

We now present the equivalent of EDHOC method 3, which is in our case, an hybridation of the previous cases. We propose two variant of this version. We start here with the first one, where the Initiator authenticates with a KEM and a signature, and the Responder with only a KEM. More specifically, this first version is a kind of hybrid between a protocol proposed in [I-D.pocero-authkem-edhoc] and one of the protocols we have proposed above.

3.4.1. Protocol overview

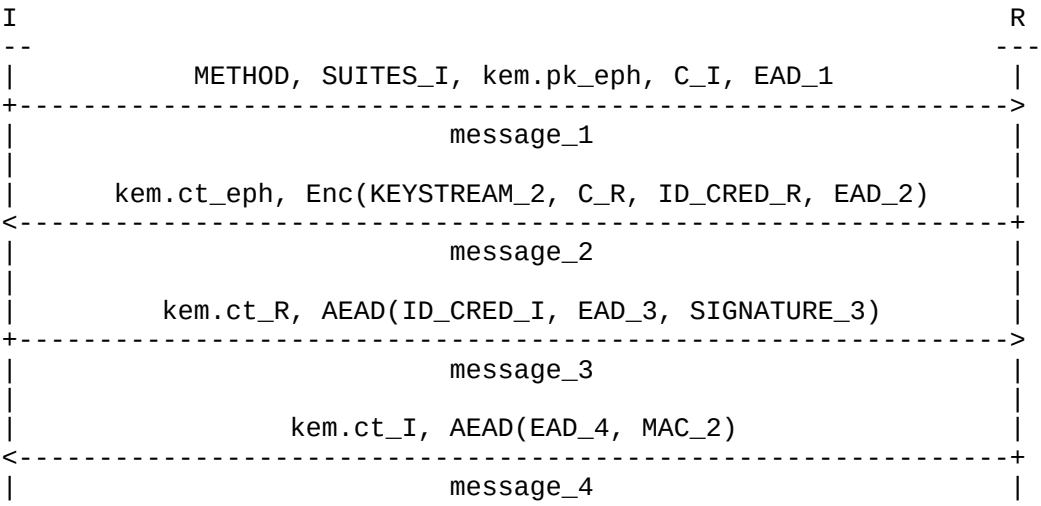


Figure 7: PQ-EDHOC, I KEM & Signs - R KEM message flow.

3.4.2. Protocol description

3.4.2.1. Formatting and sending message_1

As for our previous protocol version, the first message (message_1) consists of:

* METHOD --> as specified in [RFC9528] it is an integer specifying the authentication method the Initiator wants to use;

* SUITES_I --> it consists of an ordered set of algorithms supported by the Initiator and formatted as specified in [RFC9528];

* C_I (and also as C_R, which will appears later) --> the Connection Identifiers chosen by the Initiator (C_I) and by the Responder (C_R) as specified in [RFC9528];

* EAD_1 (and also EAD_2, EAD_3 and EAD_4, which will appear later) -> External Authorization Data, respectively included in message_1, message_2, message_3 (and optionally) message_4, and formatted as specified in [RFC9528];

* kem.pk_eph --> the Ephemeral KEM public key generated by the Initiator.

3.4.2.2. Processing message_1, formatting and sending message_2

On the reception of the first message, the Responder proceeds as in the original EDHOC protocol with elements METHOD, SUITES_I, C_I and EAD_1. In a second step, using kem.pk_eph and the KEM.Encapsulation algorithm, he computes the ephemeral ciphertext kem.ct_eph and the ephemeral shared-secret ss_eph.

The Responder selects its Connection Identifier C_R as specified in [RFC9528]. He then computes:

* TH_2 = H(kem.ct_eph, H(message_1));

* PRK_2e = EDHOC_Extract(TH_2, ss_eph);

* KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length).

In this version the Responder authenticates with the fourth mandatory message. So he directly assembles PLAINTEXT_2:

* PLAINTEXT_2 = (C_R, ID_CRED_R, TH_2, EAD_2)

then ciphers this message :

* CIPHERTEXT_2 = PLAINTEXT_2 XOR KEYSTREAM_2

and finally sends message_2 composed of CIPHERTEXT_2 and kem.ct_eph.

3.4.2.3. Processing message_2, formatting and sending message_3

On reception of the second message, the Initiator, using kemp.ct_eph, can compute the ephemeral shared-secret ss_eph. As the Responder did, he computes TH_2, PRK_2e and KEYSTREAM_2. He can now decipher and retrieve : PLAINTEXT_2 = CIPHERTEXT_2 XOR KEYSTREAM_2.

Thanks to ID_CRED_R, the Initiator obtains the long-term KEM public key of the Responder, kem.pk_R. He then uses the KEM.Encapsulation algorithm to generate ss_R and kem.ct_R.

He then computes the following elements:

* SALT_3e2m = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R);

* TH_3 = H(TH_2, PLAINTEXT_2, ID_CRED_R);

* K_3 = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);

* IV_3 = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length).

Then, in order to authenticate, he then forms PLAINTEXT_3 = (ID_CRED_I, TH_3, EAD_3) and signs it:

* SIGNATURE_3 = DS.Sign(sign.sk_I, (PLAINTEXT_3, sign_length))

where sign.sk_I is the long-term signing private key of the Initiator. Finally he sets PLAINTEXT_3A = (PLAINTEXT_3, SIGNATURE_3) and ciphers it with the AEAD encryption algorithm negociated in SUITES_I, as CIPHERTEXT_3.

The third message is then composed of:

* kem.ct_R;

* CIPHERTEXT_3.

3.4.2.4. Processing message_3, formatting and sending message_4

On reception of message_3, the Responder computes TH_3. He also used the KEM.Decapsulation algorithm with its long-term KEM private key kem.sk_R and the KEM ciphertext kem.ct_R, to obtain the shared-secret ss_R.

He can now compute SALT_3e2m, PRK_3e2m, K_3, and IV_3 as the Initiator did, and deciphers CIPHERTEXT_3 with the AEAD decryption algorithm. With PLAINTEXT_3A, he obtains kem.pk_I and sign.pk_I, and verifies the signature:

* DS.Verify(sign.pk_I, (PLAINTEXT_3, sign_length), SIGNATURE_3).

If the verification algorithm returns 1, the Initiator is properly authenticated to the Initiator. Otherwise the Responder aborts.

He then uses the KEM.Encapsulation algorithm to generate ss_I and kem.ct_I.

Following that, he computes:

* TH_4 = H(TH_3, PLAINTEXT_3A, ID_CRED_I);

* SALT_4e3m = EDHOC_KDF(PRK_3e2m, 5, TH_4, hash_length);

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I);

* K_4 = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);

* IV_4 = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length).

The Responder now needs to authenticate. For that, he computes a MAC MAC_2:

* MAC_2 = EDHOC_KDF(PRK_4e3m, 2, C_R, ID_CRED_R, TH_4, EAD_4, mac_length_2).

Using the AEAD encryption algorithm, he ciphers PLAINTEXT_4 = (EAD_4, MAC_2) and sends it to the Initiator, along with kem.ct_I. He then finishes the AKE by computing the final key PRK_out:

* PRK_out = EDHOC_KDF(PRK_4e3m, 7, TH_4, hash_length).

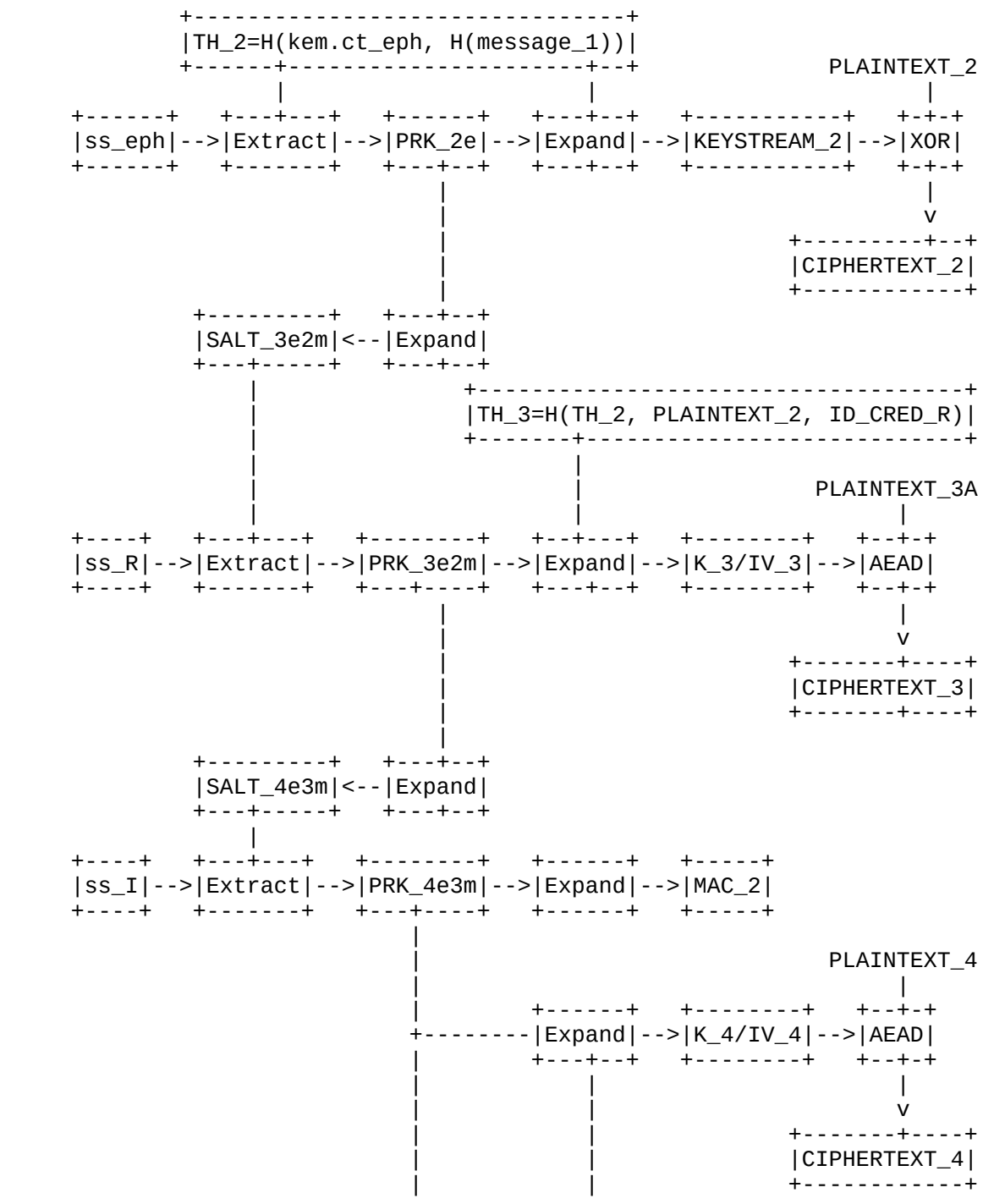
3.4.2.5. Processing message_4

In this situation the fourth message is mandatory. It is used to send the KEM ciphertext kem.ct_I to the Initiator (it cannot be shared before, since the Responder only learns the identity of the Initiator with the third message).

When receiving this last message, the Initiator computes the shared-secret ss_I thanks to the KEM.Decapsulation algorithm, and elements kem.sk_I, kem.ct_I. After that, he can compute TH_4, K_4, IV_4, and decipher CIPHERTEXT_4 thanks to the AEAD decryption algorithm. Once he had checked the well formedness of PLAINTEXT_4, he can authenticate the Responder by computing MAC_2 on its own and comparing with the one he finds in PLAINTEXT_4. If everything goes well, he finally computes PRK_out as the Responder did.

3.4.3. Associated key derivation schedule

In this section we present the key derivation schedule of the previously described protocol version.


|TH_2=H(kem.ct_eph, H(message_1))|
PLAINTEXT_2
|ss_eph|
Extract
PRK_2e
Expand
KEYSTREAM_2
XOR
CIPHERTEXT_2
SALT_3e2m
TH_3=H(TH_2, PLAINTEXT_2, ID_CRED_R)
PLAINTEXT_3A
|ss_R|
PRK_3e2m
K_3/IV_3
AEAD
CIPHERTEXT_3
SALT_4e3m
|ss_I|
PRK_4e3m
MAC_2
PLAINTEXT_4
K_4/IV_4
CIPHERTEXT_4

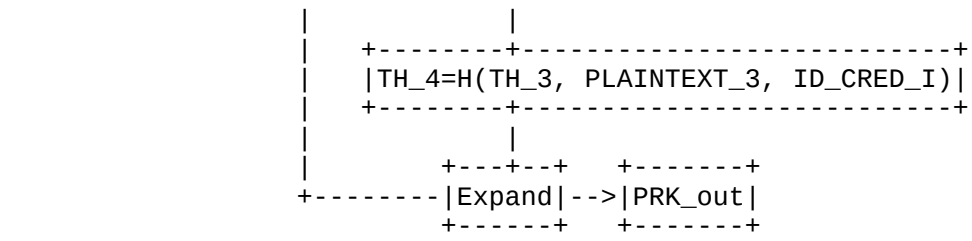


Figure 8: PQ-EDHOC, I KEM & Signs - R KEM key derivation schedule.

As above, we summarize the different computations:

```
PRK_2e      = EDHOC_Extract(TH_2, ss_eph);
KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);
SALT_3e2m   = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);
PRK_3e2m    = EDHOC_Extract(SALT_3e2m, ss_R);
K_3         = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);
IV_3        = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);
SALT_4e3m   = EDHOC_KDF(PRK_3e2m, 5, TH_4, hash_length);
PRK_4e3m    = EDHOC_Extract(SALT_4e3m, ss_I);
K_4         = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);
IV_4        = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length);
MAC_2       = EDHOC_KDF(PRK_4e3m, 2, context_2, mac_length_2)
              (with context_2 = C_R, ID_CRED_R, TH_4, EAD_4);
PRK_out     = EDHOC_KDF(PRK_4e3m, 7, TH_4, hash_length).
```

3.4.4. Additional explanations

We detail here the main elements that differ from the original EDHOC protocol.

3.4.4.1. Ephemeral KEM

As previously explain the usual ephemeral Diffie-Hellman elements are replaced by an ephemeral KEM. In this protocol, it works exactly as we previously described.

3.4.4.2. Authentication keys

For the Initiator, the authentication key MUST be: (1) a static KEM key pair (kem.sk_I, kem.pk_I) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter) *and* (2) a static signing key pair (sign.sk_I, sign.pk_I) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

For the Responder, the authentication key MUST be a static KEM key pair (kem.sk_R, kem.pk_R) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter).

3.4.4.3. Key derivation

In this version of the protocol, the keys PRK_2e, PRK_3e2m and PRK_4e3m are obtain thanks to EDHOC_Extract fonction:

* PRK_2e = EDHOC_Extract(TH_2, ss_eph) --> the salt SHALL be TH_2 and the IKM SHALL be the ephemeral shared-secret ss_eph;

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R) --> the salt SHALL be SALT_3e2m directly derived from PRK_2e, and the IKM SHALL be the ’static’ shared-secret ss_R.

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I) --> the salt SHALL be SALT_4e3m directly derived from PRK_3e2m, and the IKM SHALL be the ’static’ shared-secret ss_I.

Concerning PRK_out, here again, there is no modifications compared to the original EDHOC protocol.

3.4.5. Analysis

This variant combines our PQ-EDHOC I KEM & Signs - R Signs protocol with the version where the Initiator and Responder authenticate using a KEM, as presented in [I-D.pocero-authkem-edhoc]. The advantage of this alternative is that it preserves Responder authentication via a KEM and simply asks the Initiator to sign its identity. Additionally, there is no longer need to calculate the MAC MAC_3. In this variant, we assumed that the Initiator has a pair of static signature keys (sign.sk_I, sign.pk_I) and also a pair of static KEM keys (kem.sk_I, kem.pk_I), and that the Responder has only a pair of static KEM keys (kem.sk_R, kem.pk_R). As already explain, this is not an unusual case. From a technical point of view, messages message_1,message_2 and message_4 do not differ from those proposed in [I-D.pocero-authkem-edhoc]. Again, two major differences are worth noting regarding the messages. Firstly, here we no more have a fifth mandatory message message_5. Secondly, our third message message_3 contains an additional signature SIGNATURE_3 in PLAINTEXT_3A(befoire ciphering) (as this was the case for protocols that we have proposed above, with SIGNATURE_2 or SIGNATURE_3) (we will propose a byte analysis of this message later). Finally, from a computational point of view, both the Initiator and the Responder no longer need to calculate the MAC MAC_3. In return, here again, the Initiator must sign a message, and the Responder must verify this signature, in order to authenticate the Initiator, and send him the

KEM ciphertext kem.ct_I. Regarding the Key Derivation Schedule, aside from the IKM which adapts to post-quantum cryptographic material, we remain close to the one of EDHOC's method 3. So compared to [I-D.pocero-authkem-edhoc], we reduce the number of mandatory messages, from 5 to 4, and again we obtain a tradeoff between the size of messages, calculation capabilities, and the number of messages.

3.5. Third case: Initiator and Responder KEM and sign - version 2

We now present the second variant of the equivalent of EDHOC method 3. In this situation both the Initiator and the Responder authenticate with a KEM and a signature, which makes this protocol a hybrid of two of the protocols we presented previously.

3.5.1. Protocol overview

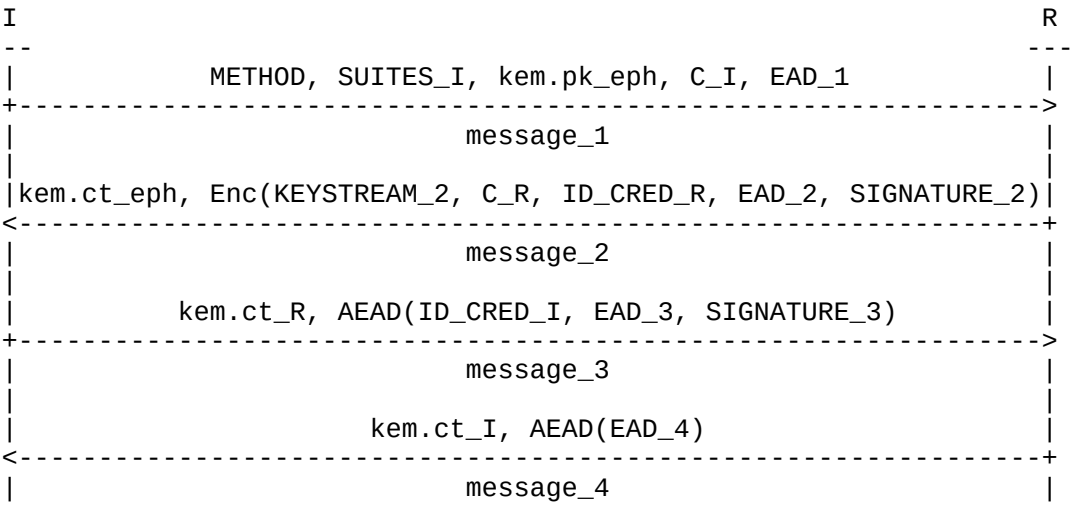


Figure 9: PQ-EDHOC, I KEM & Signs - R KEM & Signs message flow.

3.5.2. Protocol description

3.5.2.1. Formatting and sending message_1

As for our previous protocol version, the first message (message_1) consists of:

* METHOD --> as specified in [RFC9528] it is an integer specifying the authentication method the Initiator wants to use;

* SUITES_I --> it consists of an ordered set of algorithms supported by the Initiator and formatted as specified in [RFC9528];

* C_I (and also as C_R, which will appears later) --> the Connection Identifiers chosen by the Initiator (C_I) and by the Responder (C_R) as specified in [RFC9528];

* EAD_1 (and also EAD_2, EAD_3 and EAD_4, which will appear later) -> External Authorization Data, respectively included in message_1, message_2, message_3 (and optionally) message_4, and formatted as specified in [RFC9528];

* kem.pk_eph --> the Ephemeral KEM public key generated by the Initiator.

3.5.2.2. Processing message_1, formatting and sending message_2

On the reception of the first message, the Responder proceeds as in the original EDHOC protocol with elements METHOD, SUITES_I, C_I and EAD_1. In a second step, using kem.pk_eph and the KEM.Encapsulation algorithm, he computes the ephemeral ciphertext kem.ct_eph and the ephemeral shared-secret ss_eph.

The Responder selects its Connection Identifier C_R as specified in [RFC9528]. He then computes:

* TH_2 = H(kem.ct_eph, H(message_1));

* PRK_2e = EDHOC_Extract(TH_2, ss_eph);

* KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length).

In this version, the Responder authenticates with message_2 thanks to a signature. So he assembles PLAINTEXT_2:

* PLAINTEXT_2 = (C_R, ID_CRED_R, TH_2, EAD_2)

then signs this message :

* SIGNATURE_2 = DS.Sign(sign.sk_R, (PLAINTEXT_2, sign_length))

where sign.sk_R is the long-term signing private key of the Responder. And finally he ciphers PLAINTEXT_2A = (PLAINTEXT_2, SIGNATURE_2):

* CIPHERTEXT_2 = PLAINTEXT_2A XOR KEYSTREAM_2

So the second message then consists of:

* kem.ct_eph --> the ephemeral ciphertext obtained with kem.pk_eph;

* CIPHERTEXT_2.

3.5.2.3. Processing message_2, formatting and sending message_3

On reception of the second message, the Initiator, using kemp.ct_eph, can compute the ephemeral shared-secret ss_eph. As the Responder did, he computes TH_2, PRK_2e and KEYSTREAM_2. He can now decipher and retrieve : PLAINTEXT_2A = CIPHERTEXT_2 XOR KEYSTREAM_2.

Thanks to ID_CRED_R, the Initiator obtains the long-term public keys of the Responder, kemp.pk_R and sign.pk_R. At first, he checks the validity of the signature of the Responder:

* DS.Verify(sign.pk_R, (PLAINTEXT_2, sign_length), SIGNATURE_2).

If the verification algorithm returns 1, the Initiator properly authenticated the Responder. Otherwise he aborts.

Using the KEM encapsulation algorithm KEM.Encapsulation, and the long-term input material kem.pk_R of the Responder, the Initiator generated the pair (ss_R, kem.ct_R).

The shared-secret ss_R will then serve as IKM for the computation of PRK_3e2m:

* SALT_3e2m = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R).

It is now the turn of the Initiator to authenticate himself. To do so, he computes the following elements:

* TH_3 = H(TH_2, PLAINTEXT_2A, ID_CRED_R);

* K_3 = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);

* IV_3 = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);

assembles PLAINTEXT_3 = (ID_CRED_I, TH_3, EAD_3), and signs PLAINTEXT_3:

* SIGNATURE_3 = DS.Sign(sign.sk_I, (PLAINTEXT_3, sign_length))

where sign.sk_I is the long-term signing private key of the Initiator. Then, with the AEAD encryption algorithm negociated in SUITES_I, he ciphers PLAINTEXT_3A = (PLAINTEXT_3,SIGNATURE_3) as CIPHERTEXT_3.

The third message is then composed of:

* kem.ct_R;

* CIPHERTEXT_3.

3.5.2.4. Processing message_3, formatting and sending message_4

On reception of message_3, the Responder computes TH_3, K_3 and IV_3, and deciphers CIPHERTEXT_3 with the AEAD decryption algorithm. With PLAINTEXT_3A, he obtains the Initiator's long-term public keys kem.pk_I and sign.pk_I, and so he verifies the signature:

* DS.Verify(sign.pk_I, (PLAINTEXT_3, sign_length), SIGNATURE_3).

If the verification algorithm returns 1, the Initiator is properly authenticated to the Initiator. Otherwise the Responder aborts. He then uses the KEM.Encapsulation algorithm to generate ss_I and kem.ct_I.

Following that, he computes:

* TH_4 = H(TH_3, PLAINTEXT_3A, ID_CRED_I);

* SALT_4e3m = EDHOC_KDF(PRK_3e2m, 5, TH_4, hash_length);

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I);

* K_4 = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);

* IV_4 = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length).

Using the AEAD encryption algorithm, he ciphers PLAINTEXT_4 = EAD_4 and sends it to the Initiator, along with kem.ct_I. He then finishes the AKE by computing the final key PRK_out:

* PRK_out = EDHOC_KDF(PRK_3e2m, 7, TH_4, hash_length).

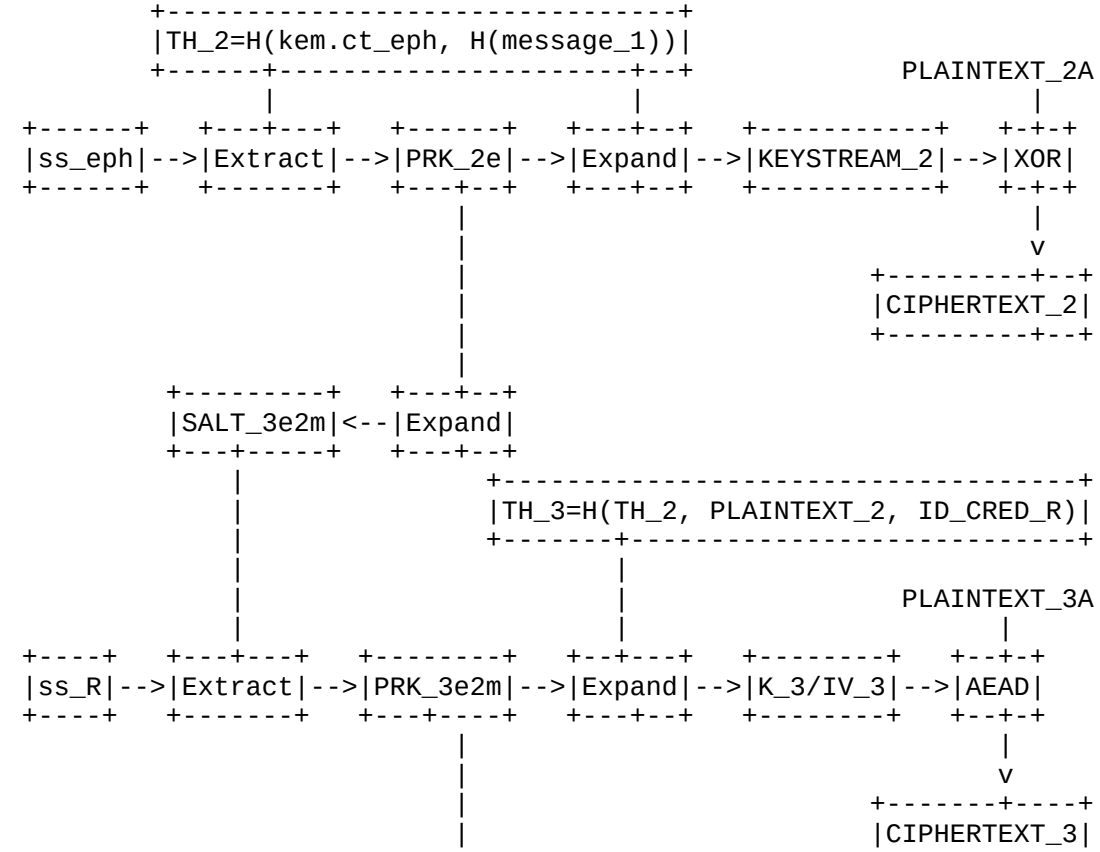
3.5.2.5. Processing message_4

In this situation the fourth message is mandatory. It is used to send the KEM ciphertext kem.ct_I to the Initiator (it cannot be shared before, since the Responder only learns the identity of the Initiator with the third message).

When receiving this last message, the Initiator computes the shared-secret ss_I thanks to the KEM.Decapsulation algorithm, and elements kem.sk_I, kem.ct_I. After that, he can compute TH_4, K_4, IV_4, and decipher CIPHERTEXT_4 thanks to the AEAD decryption algorithm. Once he had checked the well formedness of PLAINTEXT_4, he can finally compute PRK_out as the Responder did.

3.5.3. Associated key derivation schedule

In this section we present the key derivation schedule of the previously described protocol version.



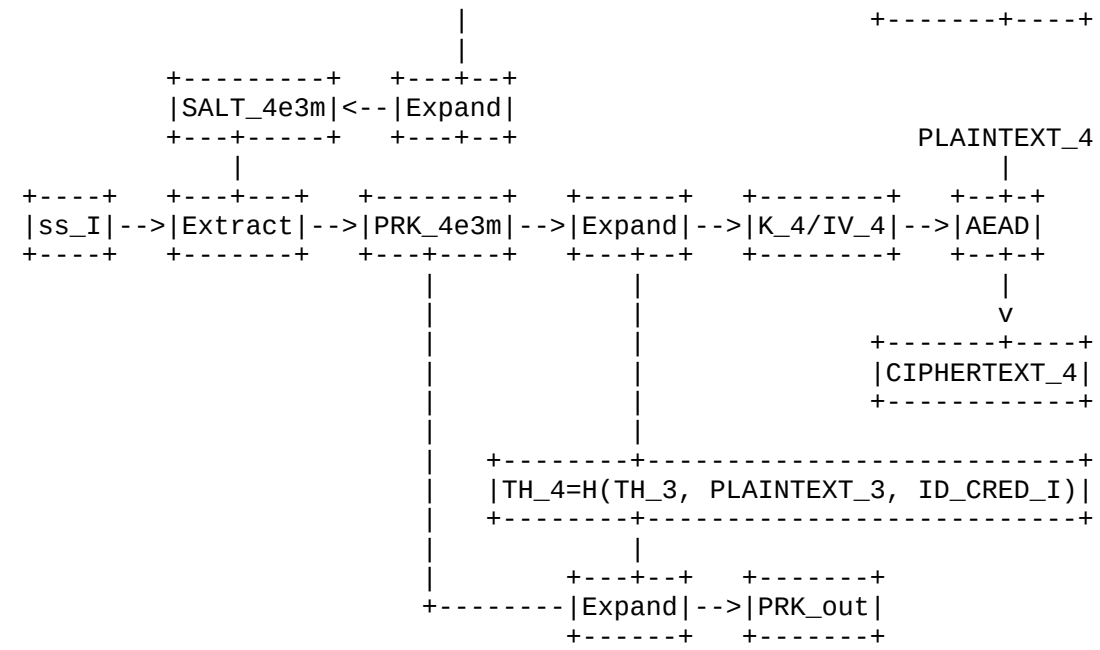


Figure 8: PQ-EDHOC, I KEM & Signs - R KEM & Sings key derivation schedule.

Here, we summarize the different computations:

```
PRK_2e      = EDHOC_Extract(TH_2, ss_eph);
KEYSTREAM_2 = EDHOC_KDF(PRK_2e, 0, TH_2, plaintext_length);
SALT_3e2m   = EDHOC_KDF(PRK_2e, 1, TH_2, hash_length);
PRK_3e2m    = EDHOC_Extract(SALT_3e2m, ss_R);
K_3         = EDHOC_KDF(PRK_3e2m, 3, TH_3, key_length);
IV_3        = EDHOC_KDF(PRK_3e2m, 4, TH_3, iv_length);
SALT_4e3m   = EDHOC_KDF(PRK_3e2m, 5, TH_4, hash_length);
PRK_4e3m    = EDHOC_Extract(SALT_4e3m, ss_I);
K_4         = EDHOC_KDF(PRK_4e3m, 8, TH_4, key_length);
IV_4        = EDHOC_KDF(PRK_4e3m, 9, TH_4, iv_length);
PRK_out     = EDHOC_KDF(PRK_4e3m, 7, TH_4, hash_length).
```

3.5.4. Additional explanations

We detail here the main elements that differ from the original EDHOC protocol.

3.5.4.1. Ephemeral KEM

As previously explain the usual ephemeral Diffie-Hellman elements are replaced by an ephemeral KEM. In this protocol, it works exactly as we previously described.

3.5.4.2. Authentication keys

For the Initiator, the authentication key MUST be: (1) a static KEM key pair (kem.sk_I, kem.pk_I) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter) *and* (2) a static signing key pair (sign.sk_I, sign.pk_I) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

For the Responder, the authentication key MUST be: (1) a static KEM key pair (kem.sk_R, kem.pk_R) generated using a KEM.KeyGen(k) algorithm (where k is a security parameter) *and* (2) a static signing key pair (sign.sk_R, sign.pk_R) generated using a DS.KeyGen(k) algorithm (where k is a security parameter).

3.5.4.3. Key derivation

In this version of the protocol, the keys PRK_2e, PRK_3e2m and PRK_4e3m are obtain thanks to EDHOC_Extract fonction:

* PRK_2e = EDHOC_Extract(TH_2, ss_eph) --> the salt SHALL be TH_2 and the IKM SHALL be the ephemeral shared-secret ss_eph;

* PRK_3e2m = EDHOC_Extract(SALT_3e2m, ss_R) --> the salt SHALL be SALT_3e2m directly derived from PRK_2e, and the IKM SHALL be the ’static’ shared-secret ss_R.

* PRK_4e3m = EDHOC_Extract(SALT_4e3m, ss_I) --> the salt SHALL be SALT_4e3m directly derived from PRK_3e2m, and the IKM SHALL be the ’static’ shared-secret ss_I.

Concerning PRK_out, here again, there is no modifications compared to the original EDHOC protocol.

3.5.5. Analysis

We conclude all these descriptions, with a last one variant which combines our PQ-EDHOC I KEM & Signs - R Signs protocol with our PQ-EDHOC I Signs - R KEM & Signs protocol. In this situation, both the Initiator and the Responder will authenticate using a signature, and there will be no longer MACs MAC_2 and MAC_3 to compute. In this variant, we assumed that the Initiator has a pair of static signature keys (sign.sk_I, sign.pk_I) and also a pair of static KEM keys

(kem.sk_I, kem.pk_I), and that the Responder has a pair of static signature keys (sign.sk_R, sign.pk_R) and also a pair of static KEM keys (kem.sk_R, kem.pk_R). As already explain, this is not an unusual case. From a technical point of view, only message message_1 remains unchanged as the one proposed in [I-D.pocero-authkem-edhoc]. Here, four major differences are worth noting regarding the messages. Firstly, here we no more have a fifth mandatory message message_5. Secondly, our second and third messages message_2 and message_3 contain each an additional signature SIGNATURE_2 and SIGNATURE_3 respectively in PLAINTEXT_2A and in PLAINTEXT_3A. Moreover, the plaintext PLAINTEXT_4 no longer includes MAC_2, which implies that the ciphertext CIPHERTEXT_4 is lighter, and so is our fourth message message_4 (we will propose a byte analysis of these messages later). Finally, from a computational point of view, both the Initiator and the Responder no longer need to calculate the MACs MAC_2 and MAC_3. In return, both the Initiator and the Responder must sign a message and verify signature, in order to authenticate themselves. This allows the Initiator to authenticate the Responder directly when he receives the second message, and vice versa for the Responder when he receives the third message. That is why we no longer need MAC_2 and MAC_3 in the fourth and fifth messages. Regarding the Key Derivation Schedule, aside from the IKM which adapts to post-quantum cryptographic material, we remain close to the one of EDHOC’s method 3. So compared to [I-D.pocero-authkem-edhoc], we reduce the number of mandatory messages, from 5 to 4, and again we obtain a tradeoff between the size of messages, calculation capabilities, and the number of messages.

4. Security Considerations

We propose here a global security analysis of our five proposals (if necessary, we number the protocols from 1 to 5 according to their order of appearance in the document). At the end of each handshake, both endpoints:

* securely compute the session key PRK_out;

* securely authenticate their partner.

For the first protocol, the Responder authenticates through the MAC in message_2, while the Initiator authenticates through the signature produced in message_3. This approach is similar to that of EDHOC [RFC9528] in method 1.

For the second protocol, the Responder’s authentication is guaranteed by its signature in message_2, while the Initiator authenticates through its signature in message_3. Note that in this protocol variant, the Responder’s signature is securely ciphered in the second message.

The third protocol is essentially a mirrored version of the second protocol, and the authentication methods of the Initiator and the Responder are switched.

For the fourth protocol, this authentication is ensured by the Responder’s MAC, sent in message_4, and the Initiator’s signature, sent encrypted in message_3.

Finally, for the fifth protocol, authentication is guaranteed by the Initiator’s and Responder’s signatures, both sent encrypted in the third and second message respectively.

Our hybrid approaches (combining KEM and signature for authentication) follow the structure of the Key Derivation Schedule in EDHOC [RFC9528]. The use of a ’static’ shared-secret in key derivation strengthens the security of the key between two non-compromised parties (e.g., if the ephemeral KEM key of the session is compromised), and to some extent also strengthens forward-security and authentication (e.g., if a long-term signature secret key is compromised).

4.1. Forward Secrecy

A forward-secrecy attacker could compromise the long-term signature key sign.sk and/or the static KEM secret key kem.sk for one of the endpoints, after that particular endpoint has completed a protocol session. The attacker’s goal is to compromise the session key PRK_out.

First note that compromising long-term signature keys does not compromise the security of past sessions.

If the static KEM secret key of the Initiator kem.sk_I or of the Responder kem.sk_R (depending on the use case and protocol) is compromised, the attacker can retrieve the shared secrets ss_I or ss_R. However, this knowledge is insufficient to learn PRK_out. This is because, as in the case of EDHOC [RFC9528] (and similarly to the recent proposals [I-D.pocero-authkem-edhoc] and [I-D.pocero-authkem-ikr-edhoc]), PRK_out is derived from all shared secrets of the session, including the ephemeral KEM secret ss_eph. As a result, to recompute PRK_out, attackers would also need to compromise the ephemeral KEM secret ss_eph used in the session. Thus, each of the five protocols achieves Forward Secrecy (FS) provided that the ephemeral KEM secret ss_eph remains uncompromised.

4.2. Identity protection

Protocol 1 is relatively close to the one proposed in [I-D.pocero-authkem-ikr-edhoc]. Concerning the Identity Protection property, in the way it is constructed, protocol 1 guarantees this property against passive attackers for the Responder and active attackers for the Initiator. Indeed, the same attacks as in EDHOC [RFC9528] by an active attacker allow learning the identity of the Responder. Moreover, since all information about identities is encrypted from the second message onwards, this ensures Identity Protection against passive/active attackers for Responder/Initiator, provided that long-term keys are not compromised. However, we want to point out a disputed aspect regarding this property. We think it might be possible to reuse the security game introduced in [CottierPointcheval23], reused in [LIEDHOC] and adapted in [EDHOCPSK] to prove Identity Protection. But this security game seems difficult to adapt to the protocol proposed in [I-D.pocero-authkem-ikr-edhoc] due to the encryption present in the first message. We suggest a more global approach for this property, possibly based on [PPAKE].

To return to a more classical definition of Identity Protection property, protocols 2 to 5 keep a similar structure as EDHOC, and so inherit of this property. In other words, the identities of the Initiator and the Responder are never sent in plaintext, always encrypted, and only from the second message onwards. Moreover, the various signatures executed by the endpoints are encrypted in the messages and therefore do not appear in clear in the exchange, as this is the case in the original EDHOC protocol.

According to the definition in [CottierPointcheval23], protocols 2 to 5 verify the Identity Protection property against passive attackers for the Responder and against active attackers for the Initiator. However, we can still keep our previous proposal in mind and try to prove this property in a more general framework for these protocols as well.

4.3. Downgrade Protection

Our protocols protect against downgrade attacks in the same way as EDHOC since the choice of methods and cipher suites is included in the calculation of certain THs and is linked to Connection Identifiers of each session (and then to endpoint identities). As a result, we have the same Downgrade Protection as in EDHOC [RFC9528].

4.4. Transcript Hash Binding

In our protocols, transcript hashes are constructed as in [RFC9528]. They contain elements from previous messages as well as endpoint identities. They are also involved in the computation of derived and intermediaries keys (as explained in the Key Derivation Schedules). Thus, like EDHOC, they provide protection against misbinding attacks (since identities are linked to keys and messages via THs) and inter-session replay attacks (since any changes to a message or injected message from another session invalidates subsequent THs). Consequently, at each step of the handshake, the integrity of the process is verified using THs.

4.5. External Authorization Data (EAD)

The External Authorization Data (EAD) in our protocols behaves similarly to what is done in EDHOC [RFC9528] and [I-D.pocero-authkem-edhoc]. The first two messages message_1 and message_2 carry EAD (partially encrypted), which are not yet authenticated. This makes them potentially vulnerable to replay attacks by an active attacker. The last two messages message_3 and message_4 protect the EAD using either MAC or signature, as well as AEAD encryption. Thus, like in EDHOC [RFC9528], our 5 protocols' EAD data must be treated as untrusted until the handshake is successfully completed.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[I-D.pocero-authkem-edhoc]
Fraile, L. P., Koulamas, C., Fournaris, A. P., and E. Haleplidis, "KEM-based Authentication for EDHOC", Work in Progress, Internet-Draft, draft-pocero-authkem-edhoc-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-pocero-authkem-edhoc-02>.

[I-D.pocero-authkem-ikr-edhoc]
Fraile, L. P., Koulamas, C., Fournaris, A. P., and E. Haleplidis, "KEM-based Authentication for EDHOC in Initiator-Known Responder (IKR) Scenarios", Work in Progress, Internet-Draft, draft-pocero-authkem-ikr-edhoc-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-pocero-authkem-ikr-edhoc-02>.

[I-D.spm-lake-pqsuites]
Selander, G. and J. P. Mattsson, "Quantum-Resistant Cipher Suites for EDHOC", Work in Progress, Internet-Draft, draft-spm-lake-pqsuites-01, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-spm-lake-pqsuites-01>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/rfc/rfc5116>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/rfc/rfc8742>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9360] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/rfc/rfc9360>.

[RFC9528] Selander, G., Preu Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

6.2. Informative References

[CottierPointcheval23] Cottier, B. and D. Pointcheval, "Security Analysis of Improved EDHOC Protocol", 2022, <https://doi.org/10.1007/978-3-031-30122-3_1>.

[EDHOCPSK] Lpez-Prez, E., Watteyne, T., Marin-Lopez, R., Onete, C., Papon, C., and M. Vuini, "Pre-Shared Key Authentication With EDHOC - The SecurityPerformance Tradeoff", 2025, <https://ieeexplore.ieee.org/document/11247913>.

[LIEDHOC] Lafourcade, P., Lpez-Prez, E., Olivier-Anclin, C., Onete, C., Papon, C., and M. Vuini, "Fine-grained, privacy-augmenting LI-compliance in the LAKE standard", 2025, <https://hal.science/hal-05126079v1>.

[PPAKE] Ramacher, S., Slamanig, D., and A. Weninger, "Privacy-Preserving Authenticated Key Exchange - Stronger Privacy and Generic Constructions", 2022, <https://eprint.iacr.org/2022/1338>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9794] Driscoll, F., Parsons, M., and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes", RFC 9794, DOI 10.17487/RFC9794, June 2025, <https://www.rfc-editor.org/rfc/rfc9794>.

Authors' Addresses

Clment Papon
XLIM UMR CNRS 7252 - Limoges University
Email: clement.papon@unilim.fr

Cristina Onete
XLIM UMR CNRS 7252 - Limoges University
Email: maria-cristina.onete@unilim.fr