LAKE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2026

E. Lopez-Perez
Inria
G. Selander
J. Preu Mattsson
Ericsson
R. Marin-Lopez
University of Murcia

7 July 2025

EDHOC Authenticated with Pre-Shared Keys (PSK)
draft-ietf-lake-edhoc-psk-04

Abstract

This document specifies a Pre-Shared Key (PSK) authentication method for the Ephemeral Diffie-Hellman Over COSE (EDHOC) key exchange protocol. The PSK method enhances computational efficiency while providing mutual authentication, ephemeral key exchange, identity protection, and quantum resistance. It is particularly suited for systems where nodes share a PSK provided out-of-band (external PSK) and enables efficient session resumption with less computational overhead when the PSK is provided from a previous EDHOC session (resumption PSK). This document details the PSK method flow, key derivation changes, message formatting, processing, and security considerations.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://lake-wg.github.io/psk/#go.draft-ietf-lake-edhoc-psk.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-lake-edhoc-psk/.

Discussion of this document takes place on the LAKE Working Group mailing list (mailto:lake@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/lake/. Subscribe at https://www.ietf.org/mailman/listinfo/lake/.

Source for this draft and an issue tracker can be found at https://github.com/lake-wg/psk.

Status of This Memo

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

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction

This document defines a Pre-Shared Key (PSK) authentication method for the Ephemeral Diffie-Hellman Over COSE (EDHOC) key exchange protocol [RFC9528]. The PSK method balances the complexity of credential distribution with computational efficiency. While symmetrical key distribution is more complex than asymmetrical approaches, PSK authentication offers greater computational efficiency compared to the methods outlined in [RFC9528]. The PSK method retains mutual authentication, asymmetric ephemeral key exchange, and identity protection established by [RFC9528].

EDHOC with PSK authentication benefits use cases where two nodes share a Pre-Shared Key (PSK) provided out-of-band (external PSK). This applies to scenarios like the Authenticated Key Management Architecture (AKMA) in mobile systems or the Peer and Authenticator in Extensible Authentication Protocol (EAP) systems. The PSK method enables the nodes to perform ephemeral key exchange, achieving Perfect Forward Secrecy (PFS). This ensures that even if the PSK is compromised, past communications remain secure against active attackers, while future communications are protected from passive attackers. Additionally, by leveraging the PSK for both authentication and key derivation, the method offers quantum resistance key exchange and authentication even when used with ECDHE.

Another key use case of PSK authentication in the EDHOC protocol is session resumption. This enables previously connected parties to quickly reestablish secure communication using pre-shared keys from a prior session, reducing the overhead associated with key exchange and asymmetric authentication. By using PSK authentication, EDHOC allows session keys to be refreshed with significantly lower computational overhead compared to public-key authentication. In this case, the PSK (resumption PSK) is provisioned after the establishment of a previous EDHOC session by using EDHOC_Exporter (resumption PSK). Therefore, the external PSK is supposed to be a long-term credential while the resumption PSK is a session key.

Section 3 provides an overview of the PSK method flow and credentials. Section 4 outlines the changes to key derivation compared to [RFC9528]. Section 5 details message formatting and processing, and Section 6 describes the usage of PSK for resumption. Section 7 defines the use of EDHOC-PSK with OSCORE. Security considerations are described in Section 8, and Section 9 outlines the IANA considerations.

2. Conventions and Definitions

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], COSE Algorithms [RFC9053], CWT and CCS [RFC8392], and the Concise Data Definition Language (CDDL) [RFC8610], which is used to express CBOR data structures.

3. Protocol

In this method, the Pre-Shared Key identifier (ID_CRED_PSK), which allows retrieval of the CRED_PSK, is sent in message_3. CRED_PSK is an authentication credential associated with the PSK. Although it may use a COSE_Key representation for compatibility, the format is not restricted to COSE_Key and can vary depending on the implementation. Through this document we will refer to the Pre-Shared Key authentication method as EDHOC-PSK.

3.1. Credentials

The Initiator and Responder are assumed to share a PSK (external PSK or resumption PSK) with good amount of entropy and the following requirements:

* Only the Initiator and the Responder have access to the PSK.

* The Responder is able to retrieve CRED_PSK and the PSK, using ID_CRED_PSK.

3.1.1. ID_CRED_PSK

ID_CRED_PSK is a COSE header map containing header parameters that can identify a pre-shared key. For example:

ID_CRED_PSK = {4 : h'0f' }; 4 = 'kid'

The purpose of ID_CRED_PSK is to facilitate retrieval of the correct PSK. It is RECOMMENDED that ID_CRED_PSK uniquely identifies the corresponding CRED_PSK, since ambiguity may require the recipient to try multiple keys.

3.1.2. CRED_PSK

CRED_PSK is an authentication credential that identifies and encapsulates the PSK. It substitutes the public-key based credentials CRED_I and CRED_R defined in [RFC9528].

While CRED_PSK may adopt encoding and representation patterns from Section 3.5.2 and Section 3.5.3 of [RFC9528], it differs fundamentally in that:

* CRED_PSK contains or identifies a symmetric key, not a public authentication key.

* Authentication is achieved implicitly via the successful use of the PSK to derive keying material and encrypt and posteriorly decrypt protected messages.

A common representation of CRED_PSK is a CBOR Web Token (CWT) or CWT Claims Set (CCS) [RFC8392] whose 'cnf' claim uses the confirmation method 'COSE_Key' to carry the PSK. An example of CRED_PSK would be:

```
{                                                  /CCS/
  2 : "mydotbot",                                  /sub/
  8 : {                                            /cnf/
    1 : {                                          /COSE_Key/
       1 : 4,                                      /kty/
       2 : h'0f',                                  /kid/
      -1 : h'50930FF462A77A3540CF546325DEA214'     /k/
    }
  }
}
```

Alternative formats for CRED_PSK MAY be used, as long as they allow the recipient to identify and retrieve the correct PSK. When the authentication credential includes only a COSE_Key (e.g., a raw key reference), it SHOULD be wrapped as a CCS by prefixing it with a 'cnf' claim. In this case, the resulting CRED_PSK contains no identity beyond the key itself.

Implementations MUST ensure that CRED_PSK values used by the Initiator and the Responder carry distinct identities in their sub claims (e.g., "42-50-31-FF-EF-37-32-39" and "23-11-58-AA-B3-7F-10"). This enables correct identification of parties and prevents misbinding attacks, as per Appendix D.2 of [RFC9528]. Note that sub is used solely for identity binding and MUST NOT be included in the key derivation function, such as EDHOC_Extract(), where only the cryptographic key material (e.g., cnf) is used.

3.1.3. Encoding and processing guidelines

The following guidelines apply to the encoding and handling of CRED_PSK and ID_CRED_PSK.

* If CRED_PSK is CBOR-encoded, it SHOULD use deterministic encoding as specified in Sections 4.2.1 and 4.2.2. of [RFC8949]. This ensures consistent identification and avoids interoperability issues due to non-deterministic CBOR variants.

* If CRED_PSK is provisioned out-of-band and transported by value, it SHOULD be used as-is without re-encoding. Re-encoding might cause mismatches when comparing identifiers such as hash values or 'kid' references.

* ID_CRED_PSK SHOULD uniquely identify the correspoding CRED_PSK to avoid ambiguity. In cases where ID_CRED_PSK is a reference to a key identifier, care must be taken to ensure that 'kid' is globally unique for the PSK.

* When ID_CRED_PSK consists solely of a 'kid' parameter (i.e., { 4 : kid }), the compact encoding optimization defined in Section 3.5.3.2 of [RFC9528] MUST be applied in plaintext fields (such as PLAINTEXT_3A). For example:

  - { 4 : h'0f' } encoded as h'0f' (CBOR byte string)

  - { 4 : 21 } encoded as 0x15 (CBOR integer)

  These optimizations MUST NOT be applied in COSE header parameters or other contexts where full map structure is required.

* To mitigate misbinding attacks, identity information such as a 'sub' (subject) claim MUST be included in CRED_PSK. If no such identity is present, the authentication credential binds only to the key.

* Guidelines in [RFC9528] related to certificate chains, X.509 DER encoding, or public key validation do not apply to CRED_PSK, since it encapsulates a symmetric key and is not used for explicit signature or certificate validation.

3.2. Message Flow of EDHOC-PSK

The ID_CRED_PSK is sent in message_3, encrypted using a key derived from the ephemeral shared secret, G_XY, calculated from the Initiator's ephemeral public key G_X or from the encapsulation G_Y and the private key corresponding to G_X. When ECDHE is used, the encapsulation G_Y is the Responder's ephemeral public key. The Responder authenticates the Initiator first. Figure 1 shows the message flow of the EDHOC-PSK authentication method.

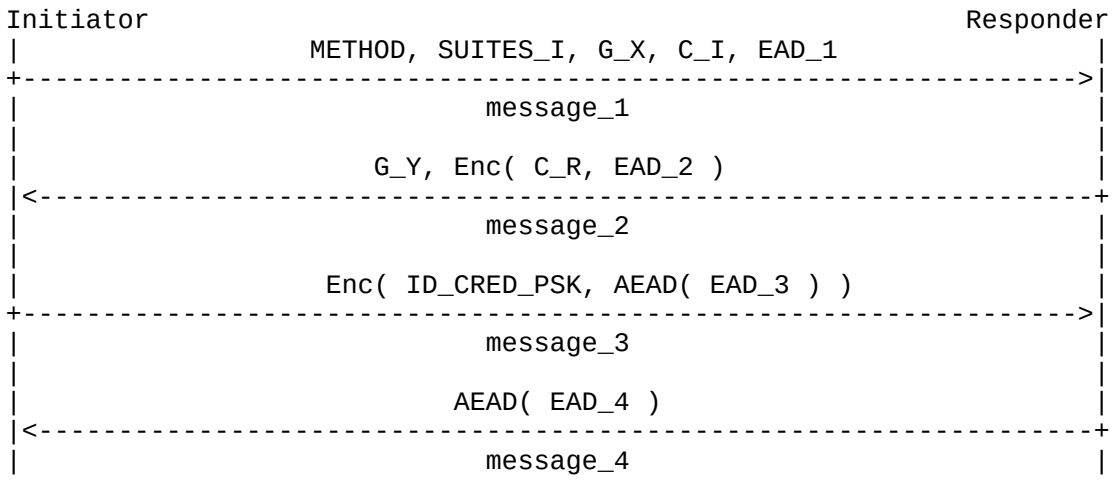

Figure 1: Overview of Message Flow of EDHOC-PSK.

This approach provides identity protection against passive attackers for both Initiator and Responder. message_4 remains optional, but is needed to authenticate the Responder and achieve mutual authentication in EDHOC if not relying on external applications, such as OSCORE. With this fourth message, the protocol achieves both explicit key confirmation and mutual authentication.

4. Key Derivation

The pseudorandom keys (PRKs) used for the PSK authentication method in EDHOC are derived using EDHOC_Extract, as done in [RFC9528]. EDHOC_Extract allows to derive fixed-length uniformly pseudorandom keys (PRKs) from ECDH shared secrets.

PRK = EDHOC_Extract(salt, IKM)

where salt and input keying material (IKM) are defined for each key. The definition of EDHOC_Extract depends on the EDHOC hash algorithm selected in the cipher suite, see Section 4.1.1 of [RFC9528].

To maintain a uniform key schedule across the different EDHOC authentication methods, the same pseudorandom key notation (PRK_2e, PRK_3e2m, and PRK_4e3m) is retained. In the case of EDHOC with PSK-based authentication, no MACs are used. Consequently, while the naming convention suggests involvement in both encryption and MAC operations, only the parts related to encryption (2e, 3e, and 4e) are relevant. The index notation does not fully reflect the functional role of the keys in this method but is preserved for consistency with other EDHOC authentication variants

The transcript hash TH_2 = H(G_Y, H(message_1)) is defined as in Section 5.3.2 of [RFC9528], the others are modified as specified below.

Figure 2 lists the key derivations that differ from those specified in Section 4.1.2 of [RFC9528]. The following data is used as in [RFC9528]:

* PRK_2e is extracted with

  - salt = TH_2, and

  - IKM = G_XY.

* SALT_4e3m is derived from PRK_3e2m and TH_3, see Figure 6 of [RFC9528].

```
PRK_3e2m     = PRK_2e
PRK_4e3m     = EDHOC_Extract( SALT_4e3m, CRED_PSK )
KEYSTREAM_3A = EDHOC_KDF( PRK_3e2m, 11, TH_3, plaintext_length_3a )
KEYSTREAM_2A = EDHOC_KDF( PRK_2e,   0, TH_2,  plaintext_length_2a )
K_3          = EDHOC_KDF( PRK_4e3m, 3, TH_3, key_length )
IV_3         = EDHOC_KDF( PRK_4e3m, 4, TH_3, iv_length )
```

Figure 2: Key Derivation of EDHOC-PSK.

where:

* KEYSTREAM_3A is used to encrypt the PLAINTEXT_3A, a concatenation of ID_CRED_PSK and CIPHERTEXT_3B, in message_3.

* KEYSTREAM_2A is used to encrypt PLAINTEXT_2A in message_2.

* TH_3 = H(TH_2, PLAINTEXT_2A).

* plaintext_length_2a is the length of PLAINTEXT_2A in message_2.

* plaintext_length_3a is the length of PLAINTEXT_3A in message_3.

Additionally, the definition of the transcript hash TH_4 is modified as:

* TH_4 = H(TH_3, ID_CRED_PSK, PLAINTEXT_3B, CRED_PSK)

5. Message Formatting and Processing

This section specifies the differences in message formatting and processing compared to Section 5 of [RFC9528].

5.1. Message 1

Message 1 is formatted and processed as specified in Section 5.2 of [RFC9528].

5.2. Message 2

5.2.1. Formatting of Message 2

Message 2 is formatted as specified in Section 5.3.1 of [RFC9528].

5.2.2. Responder Composition of Message 2

CIPHERTEXT_2A is calculated with a binary additive stream cipher, using a keystream generated with EDHOC_Expand, and the following plaintext:

* PLAINTEXT_2A = (C_R, ? EAD_2)

* CIPHERTEXT_2A = PLAINTEXT_2A XOR KEYSTREAM_2A

Contrary to [RFC9528], ID_CRED_R, MAC_2, and Signature_or_MAC_2 are not used. C_R, EAD_2 are defined in Section 5.3.2 of [RFC9528]. KESYTREAM_2A is defined in Section 4.

5.2.3. Initiator Processing of Message 2

Upon receiving message_2, the Initiator processes it as follows:

* It computes KEYSTREAM_2A, following Section 4, where plaintext_length_2a is the length of PLAINTEXT_2A.

* It decrypts CIPHERTEXT_2A using binary XOR, i.e., PLAINTEXT_2A = CIPHERTEXT_2A XOR KEYSTREAM_2A

Compared to Section 5.3.3 of [RFC9528], ID_CRED_R is not made available to the application in step 4, and steps 5 and 6 are skipped

5.3. Message 3

5.3.1. Formatting of Message 3

Message 3 is formatted as specified in Section 5.4.1 of [RFC9528].

5.3.2. Initiator Composition of Message 3

* CIPHERTEXT_3A is calculated with a binary additive stream cipher, using a keystream generated with EDHOC_Expand, and the following plaintext:

  - PLAINTEXT_3A = (ID_CRED_PSK / bstr / -24..23, CIPHERTEXT_3B)

    o If ID_CRED_PSK contains a single 'kid' parameter, i.e., ID_CRED_PSK = { 4 : kid_PSK }, then the compact encoding is applied, see Section 3.5.3.2 of [RFC9528].

    o If the length of PLAINTEXT_3A exceeds the output of EDHOC_KDF, then Appendix G of [RFC9528] applies.

  - Compute KEYSTREAM_3A as in Section 4, where plaintext_length_3a is the length of PLAINTEXT_3A.

  - CIPHERTEXT_3A = PLAINTEXT_3A XOR KEYSTREAM_3A

* CIPHERTEXT_3B is the 'ciphertext' of COSE_Encrypt0 object as defined in Section 5.2 and Section 5.3 of [RFC9528], with the EDHOC AEAD algorithm of the selected cipher suite, using the encryption key K_3, the initialization vector IV_3 (if used by the AEAD algorithm), the parameters described in Section 5.2 of [RFC9528], plaintext PLAINTEXT_3B and the following parameters as input:

  - protected = h''

  - external_aad = << ID_CRED_PSK, TH_3, CRED_PSK >>

  - K_3 and IV_3 as defined in Section 4

  - PLAINTEXT_3B = (? EAD_3)

The Initiator computes TH_4 = H(TH_3, ID_CRED_PSK, PLAINTEXT_3B, CRED_PSK), defined in Section 4.

There is no need for MAC_3 or signature, since AEAD's built-in integrity and the use of PSK-based key derivation provides implicit authentication of the Initiator.

5.3.3. Responder Processing of Message 3

Upon receiving message_3, the Responder performs the following steps:

* Derive the decryption key K_3 and IV_3 as defined in Section 4.

* Parse the structure of message_3, which consists of a stream-cipher encrypted structure, CIPHERTEXT_3A = PLAINTEXT_3A XOR KEYSTREAM_3A, where PLAINTEXT_3A = (ID_CRED_PSK, CIPHERTEXT_3B) and CIPHERTEXT_3B is the inner AEAD-encrypted object.

* Generate KEYSTREAM_3A with the same method the initiator used.

* Decrypt CIPHERTEXT_3A using binary XOR with KEYSTREAM_3A, resulting in PLAINTEXT_3A = (ID_CRED_PSK, CIPHERTEXT_3B).

* Validate or match ID_CRED_PSK to identify which PSK to use. If the ID is unrecognized, the Responder aborts.

* AEAD-decrypt CIPHERTEXT_3B using:

  - K_3, IV_3

  - external_aad = << ID_CRED_PSK, TH_3, CRED_PSK >>

  - protected = h''

  - AEAD algorithm from cipher suite

If verification fails, this indicates either the Initiator does not know the correct PSK, or the message was tampered with. Contrary, if verification succeeds, the Responder conlcudes that the Initiator knows the correct PSK, has correctly derived the transcript hash TH_3 and is actively participating in the protocol.

Lastly, the Responder computes TH_4 as defined in Section 4

No MAC_3 or signature is needed, as the AEAD tag guarantees both integrity and authenticity in this symmetric setting.

5.4. Message 4

Message 4 is formatted and processed as specified in Section 5.5 of [RFC9528].

Compared to [RFC9528], a fourth message does not only provide key confirmation but also Responder authentication. To authenticate the Responder and achieve mutual authentication, a fourth message is mandatory.

After verifying message_4, the Initiator is assured that the Responder has calculated the key PRK_out (key confirmation) and that no other party can derive the key. The Initiator MUST NOT persistently store PRK_out or application keys until the Initiator

has verified message_4 or a message protected with a derived application key, such as an OSCORE message, from the Responder and the application has authenticated the Responder.

6. PSK usage for Session Resumption

This section defines how PSKs are used for session resumption in EDHOC. Following Section 4.2 of [RFC9528], EDHOC_Exporter can be used to derive both rPSK and rID_CRED_PSK:

```
rPSK = EDHOC_Exporter( 2, h'', resumption_psk_length )
rID_CRED_PSK = EDHOC_Exporter( 3, h'', id_cred_psk_length )
```

where:

* resumption_psk_length is by default the key_length (length of the encryption key of the EDHOC AEAD algorithm of the selected cipher suite) of the session in which the EDHOC_Exporter is called.

* id_cred_psk_length is by default 2.

A peer that has successfully completed an EDHOC session, regardless of the used authentication method, MUST generate a resumption key to use for the next resumption in the present "session series", as long as it supports PSK resumption. To guarantee that both peers share the same resumption key, when a session is run using rPSK_i as a resumption key:

* The Initiator can delete rPSK_i after having successfully verified EDHOC message_4. In this case, the Responder will have derived the next rPSK_(i+1), which the Initiator can know for sure, upon receiving EDHOC message_4.

* The Responder can delete rPSK_(i-1), if any, after having successfully sent EDHOC message_4. That is, upon receiving EDHOC message_3, the Responder knows for sure that the other peer did derive rPSK_i at the end of the previous session in the "session series", thus making it safe to delete the previous resumption key rPSK_(i-1).

6.1. Cipher Suite Requirements for Resumption

When using a resumption PSK derived from a previous EDHOC exchange:

1. The resumption PSK MUST only be used with the same cipher suite that was used in the original EDHOC exchange, or with a cipher suite that provides equal or higher security guarantees.

2. Implementations SHOULD maintain a mapping between the resumption PSK and its originating cipher suite to enforce this requirement.

3. If a resumption PSK is offered with a cipher suite different from the one used in the original EDHOC session, the recipient can fail the present EDHOC session according to application-specific policies.

6.2. Privacy Considerations for Resumption

When using resumption PSKs:

* The same ID_CRED_PSK is reused each time EDHOC is executed with a specific resumption PSK.

* To prevent long-term tracking, implementations SHOULD periodically initiate a full EDHOC exchange to generate a new resumption PSK and corresponding ID_CRED_PSK. Alternatively, as stated in Appendix H of [RFC9528], EDHOC_KeyUpdate can be used to derive a new PRK_out, and consequently a new CRED_PSK and ID_CRED_PSK for session resumption.

6.3. Security Considerations for Resumption

* Resumption PSKs MUST NOT be used for purposes other than EDHOC session resumption.

* Resumption PSKs MUST be securely stored with the same level of protection as the original session keys.

* Parties SHOULD implement mechanisms to detect and prevent excessive reuse of the same resumption PSK.

7. EDHOC PSK and OSCORE

When PSK authenitcation is used and the Initiator is able to derive PRK_out before sending message_3, then the optimization described in Section 3 of [RFC9668] applies. In this scenario, the Initiator MAY concatenate EDHOC message_3 and the first OSCORE request in a single CoAP message.

8. Security Considerations

The EDHOC-PSK authentication method introduces changes with respect to the current specification of EDHOC [RFC9528]. This section analyzes the security implications of these changes.

8.1. Identity protection

EDHOC-PSK encrypts ID_CRED_PSK in message 3 with a keystream derived from the ephemeral shared secret G_XY. As a consequence, contrary to the current EDHOC methods that protect the Initiator's identity against active attackers and the Responder's identity against passive attackers (See Section 9.1 of [RFC9528]), EDHOC-PSK provides identity protection for both the Initiator and the Responder against passive attackers.

In symmetric key setups, using the same CRED_PSK for both parties or omitting role identity (e.g., sub) makes the protocol vulnerable to reflection or Selfie attacks. Separate identities in sub serve as non-cryptographic role binders and MUST be distinct.

8.2. Mutual Authentication

EDHOC-PSK provides mutual authentication, assuming the PSK remains secret. However, if the optional fourth message (message_4) is omitted, mutual authentication is not guaranteed—unless the Responder is later authenticated through an OSCORE message or other application data demonstrating possession of the PSK. When message_4 is included, the protocol ensures both mutual authentication and explicit key confirmation.

8.3. External Authorization Data Protection

Similarly to [RFC9528], EDHOC-PSK provides external authorization data protection. The integrity and confidentiality of EAD fields follow the same security guarantees as in the original EDHOC specification.

8.4. Post Quantum Considerations

Recent advancements in quantum computing suggest that the development of a Cryptographically Relevant Quantum Computer (CRQC) is likely feasible long-term. If realized, such a machine would render many currently deployed asymmetric cryptographic algorithms—such as Elliptic Curve Diffie-Hellman (ECDH)—insecure.

By leveraging a symmetric PSK for both authentication and key derivation, EDHOC-PSK provides quantum-resistant key exchange and authentication, even when used with ECDHE. However, if a cryptographically relevant quantum computer (CRQC) is realized, the ECDHE component would be broken and contribute only randomness. Consequently, EDHOC-PSK with ECDHE does not offer identity protection or Perfect Forward Secrecy (PFS) against quantum-capable adversaries. If the PSK is compromised, a passive quantum attacker could decrypt

both past and future sessions. In contrast, EDHOC-PSK combined with a quantum-resistant Key Encapsulation Mechanism (KEM), such as ML-KEM, provides identity protection and PFS even in the presence of a quantum attacker.

8.5. Independence of Session Keys

NIST mandates that an ephemeral private key shall be used in exactly one key-establishment transaction (see Section 5.6.3.3 of [SP-800-56A]). This requirement is essential for preserving session key independence and ensuring forward secrecy. The EDHOC-PSK protocol complies with this NIST requirement.

In other protocols, the reuse of ephemeral keys, particularly when combined with implementation flaws such as the absence of public key validation, has resulted in critical security vulnerabilities. Such weaknesses have allowed attackers to recover the so called “ephemeral” private key from a compromised session, thereby enabling them to compromise the security of both past and future sessions between legitimate parties. Assuming breach and minimizing the impact of compromise are fundamental zero-trust principles.

8.6. Unified Approach and Recommendations

For use cases involving the transmission of application data, application data can be sent concurrently with message_3, maintaining the protocol’s efficiency. In applications such as EAP-EDHOC, where application data is not sent, message_4 is mandatory. Thus, the EDHOC-PSK authentication method does not include any extra messages. Other implementations may continue using OSCORE in place of EDHOC message_4, with a required change in the protocol’s language to: The Initiator SHALL NOT persistently store PRK_out or application keys until the Initiator has verified message_4 or a message protected with a derived application key, such as an OSCORE message.

This change ensures that key materials are only stored once their integrity and authenticity are confirmed, thereby enhancing privacy by preventing early storage of potentially compromised keys.

Lastly, whether the Initiator or Responder authenticates first is not relevant when using symmetric keys. This consideration was important for the privacy properties when using asymmetric authentication but is not significant in the context of symmetric key usage.

9. IANA Considerations

This document has IANA actions.

9.1. EDHOC Method Type Registry

IANA is requested to register the following entry in the "EDHOC Method Type" registry under the group name "Ephemeral Diffie-Hellman Over OCSE (EDHOC)".

Value	Initiator Authentication Key	Responder Authentication Key
4	PSK	PSK

Table 1: Addition to the EDHOC Method Type Registry.

9.2. EDHOC Exporter Label Registry

IANA is requested to register the following entry in the "EDHOC Exporter Label" registry under the group name "Ephemeral Diffie-Hellman Over OCSE (EDHOC)".

Label	Description	Change Controller	Reference
2	Resumption CRED_PSK	IETF	Section 7
3	Resumption ID_CRED_PSK	IETF	Section 7

Table 2: Additions to the EDHOC Exporter Label Registry.

10. Normative References

[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>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[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>.

[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>.

[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>.

[RFC9668] Palombini, F., Tiloca, M., Hglund, R., Hristozov, S., and G. Selander, "Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)", RFC 9668, DOI 10.17487/RFC9668, November 2024, <https://www.rfc-editor.org/rfc/rfc9668>.

[SP-800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 3, April 2018, <https://doi.org/10.6028/NIST.SP.800-56Ar3>.

Appendix A. CDDL Definitions

This section compiles the CDDL definitions for easy reference, incorporating errata filed against [RFC9528].

```
   suites = [ 2* int ] / int

   ead = (
     ead_label : int,
     ? ead_value : bstr,
   )

   EAD_1 = (1* ead)
   EAD_2 = (1* ead)
   EAD_3 = (1* ead)
   EAD_4 = (1* ead)

   message_1 = (
     METHOD : int,
     SUITES_I : suites,
     G_X : bstr,
     C_I : bstr / -24..23,
     ? EAD_1,
   )

   message_2 = (
     G_Y_CIPHERTEXT_2 : bstr,
   )

   PLAINTEXT_2A = (
     C_R : bstr / -24..23,
     ? EAD_2,
   )

   message_3 = (
     CIPHERTEXT_3 : bstr,
   )

   PLAINTEXT_3A = (
     ID_CRED_PSK : header_map / bstr / -24..23,
     CIPHERTEXT_3B : bstr,
   )

   PLAINTEXT_3B = (
     ? EAD_3
   )

   message_4 = (
     CIPHERTEXT_4 : bstr,
   )

   PLAINTEXT_4 = (
     ? EAD_4,
```

```
)

error = (
  ERR_CODE : int,
  ERR_INFO : any,
)

info = (
  info_label : int,
  context : bstr,
  length : uint,
)
```

Appendix B. Test Vectors

B.1. message_1

Both endpoints are authenticated with Pre-Shared Keys (METHOD = 4)

METHOD (CBOR Data Item) (1 byte)
04

The initiator selects cipher suite 0. A single cipher suite is encoded as an int:

SUITES_I (CBOR Data Item) (1 byte)
00

The Initiator creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Initiator’s ephemeral private key
X (Raw Value) (32 bytes)
09 97 2D FE F1 EA AB 92 6E C9 6E 80 05 FE D2 9F 70 FF BF 4E
36 1C 3A 06 1A 7A CD B5 17 0C 10 E5

Initiator’s ephemeral public key
G_X (Raw Value) (32 bytes)
7E C6 81 02 94 06 02 AA B5 48 53 9B F4 2A 35 99 2D 95 72 49
EB 7F 18 88 40 6D 17 8A 04 C9 12 DB

The Initiator selects its connection identifier C_I to be the byte string 0xA, which is encoded as 0xA since it is represented by the 1-byte CBOR int 10:

Connection identifier chosen by the Initiator
C_I (CBOR Data Item) (1 byte)
0A

No external authorization data

EAD_1 (CBOR Sequence) (0 bytes)

The Initiator constructs message_1:

message_1 (CBOR Sequence) (37 bytes)
04 02 58 20 7E C6 81 02 94 06 02 AA B5 48 53 9B F4 2A 35 99
2D 95 72 49 EB 7F 18 88 40 6D 17 8A 04 C9 12 DB 0A

B.2. message_2

The Responder supports the most preferred and selected cipher suite 0, so SUITES_I is acceptable.

The Responder creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Responder’s ephemeral private key
Y (Raw Value) (32 bytes)
1E 1C 8F 2D F1 AA 71 10 B3 9F 33 BA 5E A8 DC CF 31 41 1E B3
3D 4F 9A 09 4C F6 51 92 D3 35 A7 A3

Responder’s ephemeral public key
G_Y (Raw Value) (32 bytes)
ED 15 6A 62 43 E0 AF EC 9E FB AA BC E8 42 9D 5A D5 E4 E1 C4
32 F7 6A 6E DE 8F 79 24 7B B9 7D 83

The Responder selects its connection identifier C_R to be the byte string 0x5, which is encoded as 0x5 since it is represented by the 1-byte CBOR int -10:

Connection identifier chosen by the Responder
C_R (CBOR Data Item) (1 byte)
05

The transcript hash TH_2 is calculated using the EDHOC hash algorithm: TH_2 = H(G_Y, H(message_1)), where H(message_1) is:

H(message_1) (CBOR Data Item) (32 bytes)
19 CC 2D 2A 95 7E DD 80 10 90 42 FD E6 CC 20 C2 4B 6A 34 BC
21 C6 D4 9F EA 89 5D 4C 75 92 34 0E

TH_2 (CBOR Data Item) (32 bytes)
5B 48 34 AE 63 0A 8A 0E D0 B0 C6 F3 66 42 60 4D 01 64 78 C4
BC 81 87 BB 76 4D D4 0F 2B EE 3D DE

PRK_2e is specified in Section 4.1.2 of [RFC9528]. To compute it, the Elliptic Curve Diffie-Hellman (ECDH) shared secret G_XY is needed. It is computed from G_X and Y or G_Y and X:

G_XY (Raw Value) (ECDH shared secret) (32 bytes)
2F 4A 79 9A 5A B0 C5 67 22 0C B6 72 08 E6 CF 8F 4C A5 FE 38
5D 1B 11 FD 9A 57 3D 41 60 F3 B0 B2

Then, PRK_2e is calculated as defined in Section 4.1.2 of [RFC9528]

PRK_2e (Raw Value) (32 bytes)
D0 39 D6 C3 CF 35 EC A0 CD F8 19 E3 25 79 C7 7E 1F 30 3E FC
C4 36 20 50 99 48 A9 FD 47 FB D9 29

Since the Responder authenticates using PSK, PRK_3e2m = PRK_2e.

PRK_3e2m (Raw Value) (32 bytes)
D0 39 D6 C3 CF 35 EC A0 CD F8 19 E3 25 79 C7 7E 1F 30 3E FC
C4 36 20 50 99 48 A9 FD 47 FB D9 29

No external authorization data:

EAD_2 (CBOR Sequence) (0 bytes)

The Responder constructs PLAINTEXT_2A:

PLAINTEXT_2A (CBOR Sequence) (1 byte)
05

The Responder computes KEYSTREAM_2 as defined in Section 4.1.2 of [RFC9528]

KEYSTREAM_2 (CBOR Sequence) (1 byte)
EC

The Responder calculates CIPHERTEXT_2B as XOR between PLAINTEXT_2A and KEYSTREAM_2:

CIPHERTEXT_2B (CBOR Sequence) (1 byte)
E9

The Responder constructs message_2 as defined in Section 5.3.1 of [RFC9528]:

message_2 (CBOR Sequence) (35 bytes)
58 21 ED 15 6A 62 43 E0 AF EC 9E FB AA BC E8 42 9D 5A D5 E4
E1 C4 32 F7 6A 6E DE 8F 79 24 7B B9 7D 83 E9

B.3. message_3

The Initiator computes PRK_4e3m, as described in Section 4, using SALT_4e3m and CRED_PSK:

SALT_4e3m (Raw Value) (32 bytes)
A7 C6 8F ED 7C F9 BB BC A3 83 F0 2B A4 27 45 51 EE DD 07 4A
1C F3 DB ED 4A 41 1F 33 1B 3A AA 59

CRED_PSK (Raw Value) (38 bytes)
A2 02 68 6D 79 64 6F 74 62 6F 74 08 A1 01 A3 01 04 02 41 10
20 50 50 93 0F F4 62 A7 7A 35 40 CF 54 63 25 DE A2 14

PRK_4e3m (Raw Value) (32 bytes)
6B 99 69 23 A8 81 B6 6E C4 05 AD 03 53 94 1A FB 2C DA A5 E2
AC 65 1D A0 57 9A 08 C7 86 50 2A 66

The transcript hash TH_3 is calculated using the EDHOC hash algorithm:

TH_3 = H(TH_2, PLAINTEXT_2A)

TH_3 (CBOR Data Item) (32 bytes)
BC 3B C6 10 D7 25 16 CD 70 3E F7 4C 3A 98 20 17 E5 5D 70 D2
7F 57 01 E8 91 BD 35 11 9E B9 79 88

No external authorization data:

EAD_3 (CBOR Sequence) (0 bytes)

The Initiator constructs firstly PLAINTEXT_3B as defined in Section 5.3.1.:

PLAINTEXT_3B (CBOR Sequence) (0 bytes)

It then computes CIPHERTEXT_3B as defined in Section 5.3.2. It uses ID_CRED_PSK, CRED_PSK and TH_3 as external_aad:

ID_CRED_PSK (CBOR Sequence) (1 byte)
10

CRED_PSK (Raw Value) (38 bytes)
A2 02 68 6D 79 64 6F 74 62 6F 74 08 A1 01 A3 01 04 02 41 10
20 50 50 93 0F F4 62 A7 7A 35 40 CF 54 63 25 DE A2 14

TH_3 (CBOR Data Item) (32 bytes)
A5 0B 64 8C E2 A6 23 BF C9 72 15 B2 6C 7C EE BD C4 4F 6F 79
EF AA BE 27 E8 A0 2A 25 C0 C4 93 CC

The initiator computes K_3 and IV_3

K_3 (CBOR Sequence) (16 bytes)
D8 57 B9 46 46 34 25 35 3C 3C DE BD CB 90 CB 26

IV_3 (CBOR Sequence) (13 bytes)
A1 F6 BE AF 84 64 83 2A 8D 02 C8 5F 04

It then computes CIPHERTEXT_3B:

CIPHERTEXT_3B (CBOR Sequence) (9 bytes)
48 D0 A4 37 3C 49 C1 76 9B

The Initiator computes KEYSTREAM_3 as defined in Section 4:

KEYSTREAM_3 (CBOR Sequence) ()
D0 1B 7A AD 4C AB BD 14 89 50

It then calculates PLAINTEXT_3A as stated in Section 5.3.2.:

PLAINTEXT_3A (CBOR Sequence) (10 bytes)
10 48 D0 A4 37 3C 49 C1 76 9B

It then uses KEYSTREAM_3 to derive CIPHERTEXT_3A:

CIPHERTEXT_3A (CBOR Sequence) (10 bytes)
C2 47 AA 2F 94 B8 F9 41 21 C5

The Initiator computes message_3 as defined in Section 5.3.2.:

message_3 (CBOR Sequence) (11 bytes)
4A C2 47 AA 2F 94 B8 F9 41 21 C5

The transcript hash TH_4 is calculated using the EDHOC hash algorithm: TH_4 = H(TH_3, ID_CRED_PSK, ? EAD_3, CRED_PSK)

TH_4 (CBOR Data Item) (32 bytes)
E5 67 45 39 75 66 CF F9 A6 93 DF 29 8D A4 F9 9E 1F 92 57 54
44 6B 5B 11 09 61 59 E5 C4 FC 0B FD

B.4. message_4

No external authorization data:

EAD_4 (CBOR Sequence) (0 bytes)

The Responder constructs PLAINTEXT_4:

PLAINTEXT_4 (CBOR Sequence) (0 bytes)

The Responder computes K_4 and IV_4

K_4 (CBOR Sequence) (16 bytes)
BC 34 CE B4 E3 58 36 06 A0 81 6A 53 A5 0A E1 07

IV_4 (CBOR Sequence) (13 bytes)
12 2B 58 9D EB 77 81 A1 A5 9D D4 42 7C

The Responder computes message_4:

message_4 (CBOR Sequence) (9 bytes)
48 69 C1 F1 2E 13 44 A3 93

Appendix C. Change Log

RFC Editor: Please remove this appendix.

* From -03 to -04
  - Test Vectors
  - Editorial changes
* From -02 to -03
  - Updated abstract and Introduction
  - Changed message_3 to hide the identity length from passive attackers
  - CDDL Definitions
  - Security considerations of independence of Session Keys
  - Editorial changes
* From -01 to -02
  - Changes to message_3 formatting and processing
* From -00 to -01
  - Editorial changes and corrections

Acknowledgments

The authors want to thank Christian Amsss, Scott Fluhrer, Charlie Jacomme, Marco Tiloca, and Francisco Lopez-Gomez for reviewing and commenting on intermediate versions of the draft.

This work has been partly funded by PID2023-148104OB-C43 funded by MICIU/AEI/10.13039/501100011033 (ONOFRE4), FEDER/UE EU HE CASTOR under Grant Agreement No 101167904 and EU CERTIFY under Grant Agreement No 101069471.

This work was supported partially by Vinnova - the Swedish Agency for Innovation Systems - through the EUREKA CELTIC-NEXT project CYPRESS.

Authors' Addresses

Elsa Lopez-Perez
Inria
Email: elsa.lopez-perez@inria.fr

Gran Selander
Ericsson
Email: goran.selander@ericsson.com

John Preu Mattsson
Ericsson
Email: john.mattsson@ericsson.com

Rafael Marin-Lopez
University of Murcia
Email: rafa@um.es