A Hybrid EDHOC Protocol*

SeongHan Shin[1], Adi Panca Saputra Iskandar[2], Yongho Ko[2], and Ilsun You[2] †

[1] National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
seonghan.shin@aist.go.jp
[2] Kookmin University, Seoul, South Korea
{adipancaiskandar, koyh0911, ilsunu}@gmail.com

Abstract

Recently, the Internet Engineering Task Force (IETF) Lightweight Authenticated Key Exchange (LAKE) Working Group has standardized the Ephemeral Diffie–Hellman Over COSE (EDHOC) protocol, a lightweight authenticated key exchange designed for resource-constrained Internet of Things (IoT) devices and networks. In this paper, we propose a hybrid EDHOC (Hybrid-EDHOC) protocol that achieves both classical and post-quantum security by integrating an ephemeral Post-Quantum Cryptography Key Encapsulation Mechanism (PQC–KEM) into the standard EDHOC key derivation process. The Hybrid-EDHOC protocol enables the initiator (*I*) and responder (*R*) to establish an authenticated session key while providing forward secrecy, identity protection, security against identity misbinding, and classical security against key compromise impersonation attacks. A formal security verification of Hybrid-EDHOC is conducted using ProVerif, demonstrating that the protocol ensures secrecy of cryptographic keys and application data, as well as injective agreement for the second and third message steps between *I* and *R*. For performance evaluation, 10,000 full handshake executions were conducted under identical computing conditions. Experimental results show that CPU utilization remains below 1.5% and memory usage below 3%, even in post-quantum and hybrid configurations. The classical EDHOC achieves an average latency of 3.9 ms, while the hybrid X25519 + Module-Lattice-based KEM (ML–KEM–768) variant completes in approximately 7.4 ms. These findings confirm that Hybrid-EDHOC achieves post-quantum resilience with minimal overhead (less than a two-fold increase) while maintaining lightweight and low-latency characteristics suitable for secure communication in constrained IoT environments.

Keywords: EDHOC, hybrid security, ProVerif, implementation, performance evaluation

1 Introduction

With the rapid and widespread deployment of the Internet of Things (IoT), many cryptographic protocols have been studied to meet several system constraints, such as low processing power, limited bandwidth, and restricted battery lifetime. An Authenticated Key Exchange (AKE) protocol is a core cryptographic primitive that enables two communicating peers to mutually authenticate and establish a shared session key. Recently, the Internet Engineering Task Force (IETF) Lightweight Authenticated Key Exchange (LAKE) Working Group [1] has standardized the Ephemeral Diffie–Hellman Over COSE (EDHOC) protocol [2, 3], which is a lightweight, application-layer AKE protocol tailored for resource-constrained IoT devices and networks. According to [4], EDHOC achieves a 6–14× reduction in message size, a 1.44× improvement in handshake duration, and a 2.79× reduction in energy consumption compared to Datagram Transport Layer Security version 1.3 (DTLS 1.3) [5].

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 64, December 16-18, 2025, Sapporo , Japan.

†Corresponding author

The EDHOC protocol is designed following the SIGn-and-MAc (SIGMA) family [6] of key exchange protocols, which provides a general framework for building authenticated Diffie–Hellman exchanges by combining digital signatures and Message Authentication Codes (MACs). SIGMA is widely deployed and serves as the cryptographic foundation of numerous Internet security protocols, such as the Internet Key Exchange version 2 (IKEv2) [7] and Transport Layer Security version 1.3 (TLS 1.3) [8]. To minimize message sizes between the two peers, the initiator (*I*) and responder (*R*), SIGMA introduces a MAC-then-Sign variant, where the MAC is embedded within a signature message and thus not sent separately. EDHOC adopts this SIGMA-I variant to protect the initiator's identity during authentication. In addition to signature-based authentication, EDHOC also supports static Diffie–Hellman (DH) keys (cf. the EDHOC Authentication Method Type [2]), where the digital signature is replaced by a MAC, and the MAC key is derived from an ephemeral static DH computation within the key schedule, similar to the Noise XX pattern [9]. The use of static DH keys leads to a substantial reduction in message size compared to signature-based methods.

Several studies in the literature have analyzed and extended the EDHOC protocol. Pérez *et al.* [10] summarized both symbolic and computational formal analyses (e.g., [11, 12, 13, 14, 15, 16]) of earlier EDHOC versions, discussing how previously identified vulnerabilities were mitigated to achieve the target security goals. Fraile *et al.* [17] proposed a post-quantum variant of EDHOC, termed Post-Quantum EDHOC (PQ-EDHOC), which integrates Post-Quantum Cryptography (PQC) algorithms, specifically, Key Encapsulation Mechanisms (KEMs) and digital signature schemes, and evaluated multiple PQC-KEM and PQC signature combinations with respect to execution time, energy consumption, memory usage, and network performance. Their results indicate that the Module-Lattice-based Key Encapsulation Mechanism (ML-KEM) [18, 19, 20, 21] is currently the most suitable PQC-KEM for constrained environments, whereas PQC digital signature schemes such as HAWK [22] show improved performance compared to ML-Digital Signature Algorithm (ML-DSA) [18, 23] and Fast Fourier Lattice-based Compact Signatures over NTRU (FALCON) [24], both of which are candidates for standardization under the Federal Information Processing Standards (FIPS). It is important to note that PQ-EDHOC remains based on the EDHOC Authentication Method Type 0 (Signature-Key-based Authentication) [2]. More recently, Fraile *et al.* [25] submitted an IETF Internet-Draft describing a new PQC-KEM-based authentication method for EDHOC that avoids reliance on PQC digital signatures. However, their proposal adopting an approach inspired by the Post-Quantum Noise (PQNoise) framework [26] requires substantial modifications to the original EDHOC message flow.

In this paper, we propose and implement a hybrid EDHOC (Hybrid-EDHOC) protocol that achieves both classical and post-quantum security without altering the core EDHOC message structure. Our design integrates an ephemeral PQC-KEM (specifically, ML-KEM-768) into the existing EDHOC key derivation chain alongside the classical Elliptic-Curve Diffie–Hellman (ECDH) operation using X25519, following the hybrid key combination concept defined in the IETF. We implement Hybrid-EDHOC within the standard EDHOC processing flow and evaluate its computational cost, memory usage, and latency through extensive experiments. The expected results show that the hybrid mechanism can deliver post-quantum resilience with minimal computational overhead while maintaining the lightweight, low-latency properties that make Hybrid-EDHOC suitable for constrained IoT deployments and emerging 6G edge environments.

1.1 Motivation and Our Contributions

For a global migration toward Post-Quantum Cryptography (PQC) [27, 28, 29], numerous hybrid key-exchange schemes (e.g., [30, 31, 32, 33, 34, 35]) have been proposed by both academic researchers and international standardization bodies. These hybrid schemes aim to provide dual security guarantees, classical security based on mathematical hard problems such as the discrete logarithm, and post-

quantum security based on lattice- or code-based constructions. However, to the best of our knowledge, no hybrid version of the EDHOC protocol has been defined in the literature or standardized by the IETF LAKE Working Group.

In this paper, we propose a hybrid EDHOC (Hybrid-EDHOC) protocol—a hybrid variant of EDHOC that provides both classical and post-quantum security simultaneously by integrating an ephemeral Post-Quantum Cryptography Key Encapsulation Mechanism (PQC–KEM) into the standard EDHOC key derivation process. Specifically, the proposed Hybrid-EDHOC protocol is built upon the most compact and efficient EDHOC Authentication Method Type 3 (Static Diffie–Hellman Key) [2]. Within this protocol, the initiator (I) and responder (R) establish an authenticated session key that offers forward secrecy, identity protection, security against identity misbinding, and classical security against key compromise impersonation attacks.

Our Contributions. The key contributions of this work are summarized as follows:

- We design the first Hybrid-EDHOC protocol that integrates an ephemeral PQC-KEM (specifically, the Module-Lattice-based Key Encapsulation Mechanism, ML-KEM-768) into the standard EDHOC key derivation process together with classical Elliptic-Curve Diffie–Hellman (ECDH) using X25519, following the hybrid key combination framework defined in the IETF.

- Also, we give a formal security verification of the Hybrid-EDHOC protocol in a symbolic model (ProVerif). The security verification results show that the protocol ensures secrecy of cryptographic keys and application data, as well as injective agreement for the second and third message steps between I and R. However, the first injective agreement query (Step 1) returned `false`, indicating a missing freshness or replay-prevention mechanism during the unauthenticated initiation phase. This limitation can be mitigated by incorporating a lightweight nonce, timestamp, or sequence-based freshness token into Step 1, which prevents replay and flooding attacks while preserving the lightweight design of EDHOC. For more details, see Section 4.1.

- Moreover, we implement the Hybrid-EDHOC protocol in a full software stack compatible with the current EDHOC specification, without requiring any changes to the core message formats or authentication flow. Also, we evaluate its performance through 10,000 complete handshake executions, measuring latency, Central Processing Unit (CPU) utilization, and memory consumption in both classical and hybrid configurations. The performance evaluation results demonstrate that Hybrid-EDHOC achieves post-quantum resilience with less than a two-fold increase in processing time and negligible memory overhead, thereby preserving the lightweight and low-latency characteristics required for secure communication in constrained IoT and upcoming 6G edge environments.

2 Preliminaries

Table 1 shows abbreviations and notations to be used throughout this paper.

2.1 Key Encapsulation Mechanism

A KEM (Key Encapsulation Mechanism) [36] is a triple of the following algorithms.

- **KEM.KeyGen**(pp_{KEM}): On input of a public parameter pp_{KEM}, the algorithm outputs a private-public key pair (sk_{KEM}, PK_{KEM}).

- **KEM.Enc**(PK_{KEM}): On input of a public key PK_{KEM}, the algorithm outputs a shared key k_{KEM} and a ciphertext C_{KEM}.

Table 1: Abbreviations and notations

	Meanings
I	Initiator ID
R	Responder ID
(a,A)	Static ECDH private-public key pair of I
(b,B)	Static ECDH private-public key pair of R
(x,X)	Ephemeral ECDH private-public key pair of I
(y,Y)	Ephemeral ECDH private-public key pair of R
H	Cryptographic hash function
KDF	Key derivation function (EDHOC_KDF [2])
HKDF-Extract	Key extraction function (EDHOC_Extract [2])
HKDF-Expand	Key expansion function (EDHOC_Expand [2])
Enc	Symmetric-key encryption (XOR stream cipher or AEAD)
Dec	Symmetric-key decryption (XOR stream cipher or AEAD)
TH	Transcript hash
PRK	Pseudo-random key
EK	Encryption key
IV	Initialization vector
MK	MAC key
Method	EDHOC Authentication Method Type [2]
Cipher Suites	EDHOC Cipher Suites [2] (an ordered set of algorithms for crypto agility, flexibility and modularity)
C	Connection Identifier
EAD	External Authorization Data

- **KEM.Dec**(sk_{KEM}, C_{KEM}): On input of a private key sk_{KEM} and a ciphertext C_{KEM}, the algorithm outputs a shared key k_{KEM}.

Examples of PQC-KEM include Classic McEliece KEM [37, 20, 21], FrodoKEM [38, 20, 21] and ML-KEM (CRYSTALS-Kyber) [18, 19, 20, 21].

3 A Hybrid EDHOC Protocol

In this section, we propose a hybrid EDHOC (for short, Hybrid-EDHOC) protocol that provides both classical security and post-quantum security simultaneously by instantiating KEM with PQC-KEM. Specifically, the Hybrid-EDHOC protocol is based on the most compact and efficient EDHOC Authentication Method Type 3 (Static DH Key) [2]. A main idea of the Hybrid-EDHOC protocol is that (1) static ECDH raw public keys are dealt with pre-shared secrets; (2) an ephemeral PQC-KEM is used; and (3) a PQC-KEM shared key is used as an additional key material in the computation of PRK. In the Hybrid-EDHOC protocol, initiator I and responder R share an authenticated session key PRK_{out}.

Also, the Hybrid-EDHOC protocol provides forward secrecy, identity protection, security against identity misbinding, and classical security against key compromise impersonation. Before executing the Hybrid-EDHOC protocol, I holds $((a, A \equiv g^a), (R, B))$ secretly and R stores $((b, B \equiv g^b), (I, A))$ secretly.

3.1 Step 1 (Initiator)

Initiator I executes the followings:

1. Generate an ephemeral KEM private-public key pair $(sk_{KEM}, PK_{KEM}) \leftarrow \mathbf{KEM.KeyGen}(pp_{KEM})$
2. Generate an ephemeral ECDH private-public key pair (x, X) such that $X \equiv g^x$
3. Set $M1 = (\text{Method}, \text{Cipher Suites}, X, PK_{KEM}, \text{C}_\text{I}, \text{EAD}_1)$

I sends $M1$ to R.

3.2 Step 2 (Responder)

Upon receiving $M1$ from I, responder R executes the followings:

1. Compute a KEM shared key and a KEM ciphertext $(k_{KEM}, C_{KEM}) \leftarrow \mathbf{KEM.Enc}(PK_{KEM})$
2. Generate an ephemeral ECDH private-public key pair (y, Y) such that $Y \equiv g^y$
3. Compute $TH_2 = \mathsf{H}(M1, Y, C_{KEM})$
4. Compute $PRK_{2e} = \mathsf{HKDF\text{-}Extract}(X^y, k_{KEM}, TH_2)$ and $EK_2 = \mathsf{HKDF\text{-}Expand}(PRK_{2e}, TH_2)$
5. Compute $PRK_{3e2m} = \mathsf{HKDF\text{-}Extract}(X^b, PRK_{2e})$ and $MK_2 = \mathsf{HKDF\text{-}Expand}(PRK_{3e2m}, TH_2)$
6. Compute $MAC_2 = \mathsf{KDF}(MK_2, R, B, TH_2, \text{EAD}_2, \text{len}_2)$
7. Set $msg_2 = (\text{C}_\text{R}, R, \text{EAD}_2, MAC_2)$
8. Set $M2 = (Y, C_{KEM}, \mathsf{Enc}_{EK_2}(msg_2))$

R sends $M2$ to I.

3.3 Step 3 (Initiator)

Upon receiving $M2$ from R, initiator I executes the followings:

1. Compute a KEM shared key $k_{KEM} \leftarrow \mathbf{KEM.Dec}(sk_{KEM}, C_{KEM})$
2. Compute $TH_2 = \mathsf{H}(M1, Y, C_{KEM})$
3. Compute $PRK_{2e} = \mathsf{HKDF\text{-}Extract}(Y^x, k_{KEM}, TH_2)$ and $EK_2 = \mathsf{HKDF\text{-}Expand}(PRK_{2e}, TH_2)$
4. Compute $msg_2 = \mathsf{Dec}_{EK_2}(\mathsf{Enc}_{EK_2}(msg_2))$
5. Parse $(\text{C}_\text{R}, R, \text{EAD}_2, MAC_2) \leftarrow msg_2$
6. Compute $PRK_{3e2m} = \mathsf{HKDF\text{-}Extract}(B^x, PRK_{2e})$ and $MK_2 = \mathsf{HKDF\text{-}Expand}(PRK_{3e2m}, TH_2)$
7. Verify MAC_2

8. Compute $TH_3 = \mathsf{H}(TH_2, msg_2, B)$
9. Compute $(EK_3, IV_3) = \mathsf{HKDF\text{-}Expand}(PRK_{3e2m}, TH_3)$
10. Compute $PRK_{4e3m} = \mathsf{HKDF\text{-}Extract}(Y^a, PRK_{3e2m})$ and $MK_3 = \mathsf{HKDF\text{-}Expand}(PRK_{4e3m}, TH_3)$
11. Compute $MAC_3 = \mathsf{KDF}(MK_3, I, A, TH_3, \mathrm{EAD}_3, \mathrm{len}_3)$
12. Set $msg_3 = (I, \mathrm{EAD}_3, MAC_3)$
13. Set $M3 = \mathsf{Enc}_{EK_3}(msg_3)$
14. Compute $TH_4 = \mathsf{H}(TH_3, msg_3, A)$
15. Compute a session key $PRK_{out} = \mathsf{HKDF\text{-}Expand}(PRK_{4e3m}, TH_4)$ and an application key $AK = \mathsf{HKDF\text{-}Expand}(\mathsf{HKDF\text{-}Expand}(PRK_{out}))$

I sends $M3$ to R.

3.4 Step 4 (Responder)

Upon receiving $M3$ from I, responder R executes the followings:

1. Compute $TH_3 = \mathsf{H}(TH_2, msg_2, B)$
2. Compute $(EK_3, IV_3) = \mathsf{HKDF\text{-}Expand}(PRK_{3e2m}, TH_3)$
3. Compute $msg_3 = \mathsf{Dec}_{EK_3}(\mathsf{Enc}_{EK_3}(msg_3))$
4. Parse $(I, \mathrm{EAD}_3, MAC_3) \leftarrow msg_3$
5. Compute $PRK_{4e3m} = \mathsf{HKDF\text{-}Extract}(A^y, PRK_{3e2m})$ and $MK_3 = \mathsf{HKDF\text{-}Expand}(PRK_{4e3m}, TH_3)$
6. Verify MAC_3
7. Compute $TH_4 = \mathsf{H}(TH_3, msg_3, A)$
8. Compute a session key $PRK_{out} = \mathsf{HKDF\text{-}Expand}(PRK_{4e3m}, TH_4)$ and an application key $AK = \mathsf{HKDF\text{-}Expand}(\mathsf{HKDF\text{-}Expand}(PRK_{out}))$

In the above, Enc in $M2$ is an XOR stream cipher, and Enc in $M3$ is an AEAD (Authenticated Encryption with Associated Data). Also, an optional fourth message in the EDHOC protocol is omitted in the Hybrid-EDHOC protocol. Figure 1 shows a protocol diagram and a key schedule of the Hybrid-EDHOC protocol.

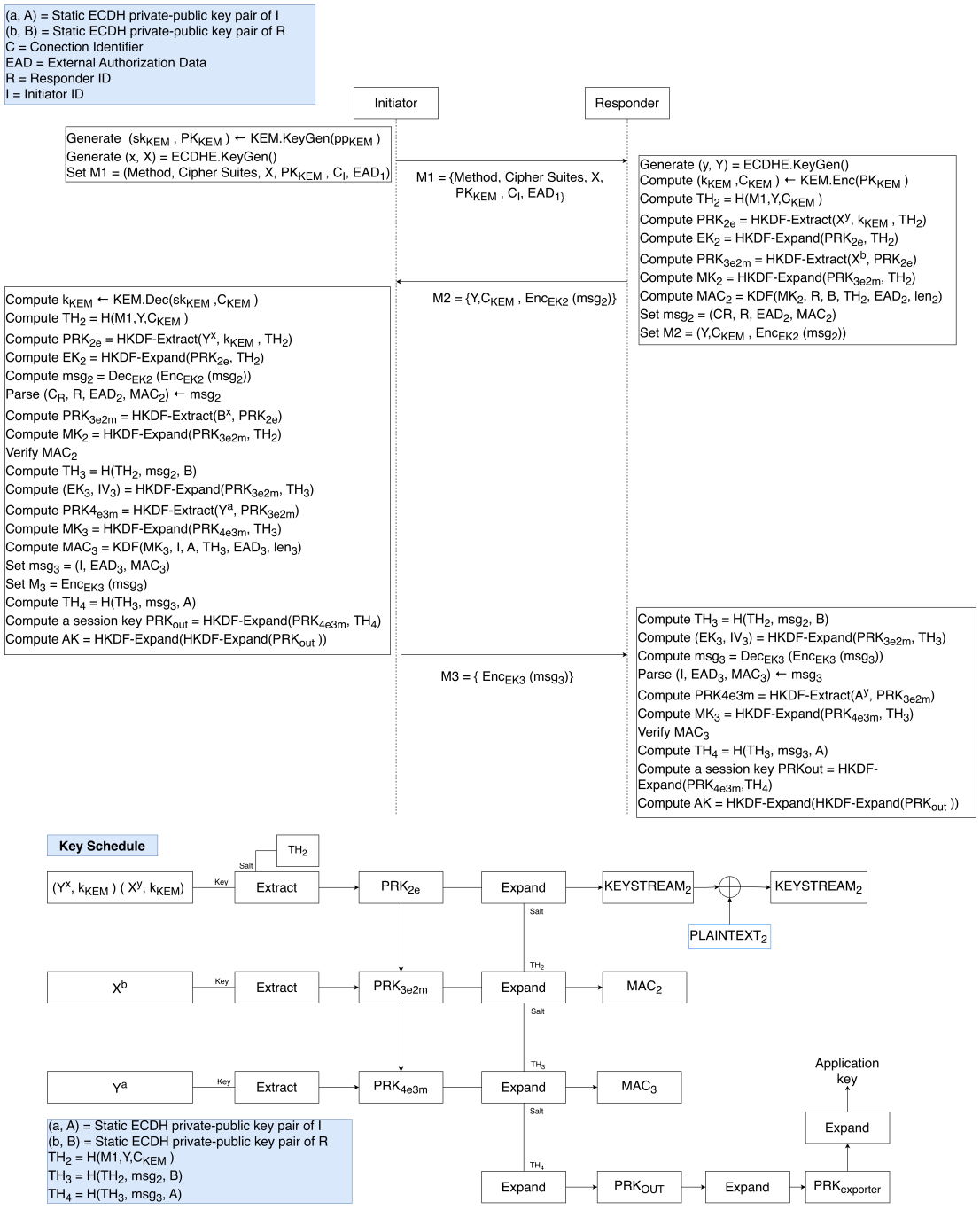

Figure 1: Protocol diagram (above) and key schedule (below)

3.5 Comparison with EDHOC

Here, we compare the EDHOC [2] and Hybrid-EDHOC protocols with respect to several classical and post-quantum security properties (mutual authentication, security of session key, forward secrecy, identity protection, security against identity misbinding, security against KCI (Key Compromise Impersonation)). We summarize comparative results in Table 2. The Hybrid-EDHOC protocol provides mutual authentication, security of session key, forward secrecy, identity protection, and security against identity misbinding since a post-quantum attacker basically cannot compute PRK_{3e2m}. Note that static ECDH raw public keys (i.e., A and B) are pre-shared secrets between initiator I and responder R. However, the Hybrid-EDHOC protocol inherently does not provide post-quantum security against KCI since its construction is based on EDHOC Authentication Method Type 3 (Static DH Key) [2]. For example, a post-quantum attacker who gets R's secrets $((b, B \equiv g^b), (I, A))$ can impersonate I by recovering the ECDH raw private key a from A.

Table 2: Security comparison of the EDHOC [2] and Hybrid-EDHOC protocols

	EDHOC [2]		Hybrid-EDHOC	
	Classical security	Post-quantum security	Classical security	Post-quantum security
Mutual authentication	Yes	No	Yes	Yes
Security of session key	Yes	No	Yes	Yes
Forward secrecy	Yes	No	Yes	Yes
Identity protection	Yes	No	Yes	Yes
Security against identity misbinding	Yes	No	Yes	Yes
Security against KCI	Yes	No	Yes	No

4 Formal Security Verification using ProVerif

This section gives a formal security verification of the Hybrid-EDHOC protocol in a symbolic model (ProVerif), declaring the abstract types, long-term secrets, and cryptographic primitives used in the analysis. Using these declarations, we encode correspondence and secrecy queries (e.g., injective agreement and key secrecy) and check whether an adversary can produce counterexamples to guide any necessary fixes.

Algorithm 1 Declaration

```
(* Type specification *)
type skey.
type spk.
type exponent.
type G.
type key.
(* Variable & Constant *)
free b: exponent [private].
free a: exponent [private].
const TAG_LEN2: bitstring.
const TAG_LEN3: bitstring.
const g: G.
(* Channel specification *)
free c: channel.
```

This block defines the cryptographic primitives in the symbolic model, including ECDH, KEM,

hash, KDF, and AEAD. It models key derivation via extract/expand rules, shared-secret generation through KEM reduction, and uses converters (g2b, k2b, b2k) to map elements into bitstrings for verification.

Algorithm 2 Function and Converter

```
(* ECDHE *)
fun exp(G, exponent): G.
equation forall x: exponent, y: exponent;
 exp(exp(g, x), y) = exp(exp(g, y), x).
(* Key Encapsulation Mechanism *)
fun pk(skey): spk.
fun kem_k(spk): key[private].
fun kem_ct(spk): key.
fun kem_dec(skey, key): key
 reduc forall sk: skey; kem_dec(sk, kem_ct(pk(sk))) = kem_k(pk(sk)).
(* Hash & key Derivation *)
fun H(bitstring): bitstring.
fun KDF(bitstring): bitstring.
fun extract(bitstring, bitstring, bitstring): key.
fun expands(key, bitstring): key.
(* Encryption & Decryption *)
fun aead_enc(key, bitstring, bitstring): bitstring.
fun aead_dec(key, bitstring, bitstring): bitstring
reduc forall k: key, p: bitstring, ad: bitstring;
  aead_dec(k, ad, aead_enc(k, ad, p)) = p.
(* Converter *)
fun g2b(G): bitstring [data, typeConverter].
fun k2b(key): bitstring [data, typeConverter].
fun b2k(bitstring): key [data, typeConverter].
```

Algorithm 3 Authentication queries, Secrecy assumptions, dan Debugger

```
(* Event *)
event S_STEP1_I_to_R(bitstring, bitstring).
event E_STEP1_I_to_R(bitstring, bitstring).
event S_STEP2_R_to_I(key, bitstring).
event E_STEP2_R_to_I(key, bitstring).
event S_STEP3_I_to_R(key, bitstring).
event E_STEP3_I_to_R(key, bitstring).
(* Security requirements verification *)
Q1 ; query k: bitstring, msg: bitstring;
inj-event(E_STEP1_I_to_R(k, msg)) ==> inj-event(S_STEP1_I_to_R(k, msg)).
Q2 ; query k: key, msg: bitstring;
inj-event(E_STEP2_R_to_I(k, msg)) ==> inj-event(S_STEP2_R_to_I(k, msg)).
Q3 ; query k: key, msg: bitstring;
inj-event(E_STEP3_I_to_R(k, msg)) ==> inj-event(S_STEP3_I_to_R(k, msg)).
(* Security requirements verification *)
free app_data1, app_data2: bitstring [private].
Q4 ; query attacker(CI).
Q5 ; query attacker(CR).
Q6 ; query attacker(skKEM).
Q7 ; query attacker(app_data1).
Q8 ; query attacker(app_data2).
```

Algorithm 4 Initiator process (complete)

Input : Initiator private key $skKEM$: skey, peer public key $PKKEM$: spk;
global params g : G, a : exponent; constants TAG_LEN2, TAG_LEN3; channel c

Output: Messages $M1$ and $M3$ sent on c; ciphertext e_app_data2; final keys $(PRKout, AK)$

begin

```
let Initiator(skKEM:skey, PKKEM:spk) =
new Method: bitstring;
new CipherSuites: bitstring;
new CI: bitstring;
new EAD1: bitstring;

new x: exponent;
let X = exp(g, x) in
let A_pub = exp(g, a) in
let M1 = (Method, CipherSuites, X, PKKEM, CI, EAD1, A_pub) in
event S_STEP1_I_to_R(CI,M1);
out(c, M1);

in(c, M2: bitstring);
let (Y: G, CKEM: key, C2: bitstring) = M2 in
let kKEM = kem_dec(skKEM, CKEM) in
let TH2 = H(( (Method, CipherSuites, X, PKKEM, CI, EAD1, A_pub),
g2b(Y), k2b(CKEM))) in
let Yx= exp(Y, x) in
let prk2e = extract(g2b(Yx), k2b(kKEM), TH2) in
let EK2 = expands(prk2e, TH2) in
let msg2 = aead_dec(EK2, TH2, C2) in
let (CR: bitstring, R: bitstring, EAD2: bitstring, MAC2: bitstring) = msg2 in
let B = exp(g, b) in
let Bx = exp(B, x) in
let prk3e2m = extract(g2b(Bx), k2b(prk2e), TH2) in
let MK2 = expands(prk3e2m, TH2) in
let len2 = H((TAG_LEN2, TH2, EAD2)) in
let mac2_check = KDF((k2b(MK2), R, g2b(B), TH2, EAD2, len2)) in

if MAC2 = mac2_check then(
event E_STEP2_R_to_I(b2k(MAC2), msg2);
  let TH3 = H((TH2, msg2, g2b(B))) in
  let EK3 = expands(prk3e2m, TH3) in
  let IV3 = H((k2b(prk3e2m), TH3)) in
  let Ya = exp(Y, a) in
  let prk4e3m = extract(g2b(Ya), k2b(prk3e2m), TH3) in
  let MK3 = expands(prk4e3m, TH3) in
  new I: bitstring;
  new EAD3: bitstring;
  let len3 = H((TAG_LEN3, TH3, EAD3)) in
  let MAC3 = KDF((k2b(MK3), I, g2b(A_pub), TH3, EAD3, len3)) in
  let msg3 = (I, EAD3, MAC3) in
  let M3 = aead_enc(EK3, TH3, msg3) in
  event S_STEP3_I_to_R(b2k(MAC3), M3);
  out(c, M3);
  let TH4 = H((TH3, msg3, g2b(A_pub))) in
  let PRKout = expands(prk4e3m, TH4) in
  let AK = expands(PRKout, TH4) in
  let e_app_data2 = aead_enc(AK, TH4, app_data2) in
  out(c, c_debug);
  out(c, e_app_data2);
  0
)else
  0
```

end

Algorithm 5 Responder process (complete)

Input : Channel c; global params g : G, exponents a, b; constants TAG_LEN2, TAG_LEN3

Output: Second-flight message $M2$ sent on c; verification of $M3$; final $(PRKout, AK)$ and ciphertext e_app_data1

```
begin
  let Responder =
  in(c, M1:bitstring);
  let (Method:bitstring, CipherSuites:bitstring, X:G, PKKEM:spk, CI:bitstring,
  EAD1:bitstring, A_pub:G) = M1 in
  event E_STEP1_I_to_R(CI, M1);
  let kKEM = kem_k(PKKEM) in
  let CKEM = kem_ct(PKKEM) in
  new y: exponent;
  let Y = exp(g, y) in
  let TH2 = H(((Method, CipherSuites, X, PKKEM, CI, EAD1, A_pub), g2b(Y), k2b(CKEM)))
  in
  let Xy = exp(X, y) in
  let prk2e = extract(g2b(Xy), k2b(kKEM), TH2) in
  let EK2 = expands(prk2e, TH2) in
  let Xb = exp(X, b) in
  let prk3e2m = extract(g2b(Xb), k2b(prk2e), TH2) in
  let MK2 = expands(prk3e2m, TH2) in
  new EAD2:bitstring;
  let len2 = H((TAG_LEN2, TH2, EAD2)) in
  let B = exp(g, b) in
  new R:bitstring;
  let MAC2 = KDF((k2b(MK2), R, g2b(B), TH2, EAD2, len2)) in
  new CR:bitstring;
  let msg2 = (CR, R, EAD2, MAC2) in
  let M2 = (Y, CKEM, aead_enc(EK2, TH2, msg2)) in
  event S_STEP2_R_to_I(b2k(MAC2), msg2);
  out(c, M2);
  in(c, M3: bitstring);
  let TH3 = H((TH2, msg2, g2b(B))) in
  let EK3 = expands(prk3e2m, TH3) in
  let IV3 = H((k2b(prk3e2m), TH3)) in
  let msg3 = aead_dec(EK3, TH3, M3) in
  let (I: bitstring, EAD3: bitstring, MAC3: bitstring) = msg3 in
  let Ay = exp(A_pub, y) in
  let prk4e3m = extract(g2b(Ay), k2b(prk3e2m), TH3) in
  let MK3 = expands(prk4e3m, TH3) in
  let len3 = H((TAG_LEN3, TH3, EAD3)) in
  let mac3_check = KDF((k2b(MK3), I, g2b(A_pub), TH3, EAD3, len3)) in
  if MAC3 = mac3_check then
    event E_STEP3_I_to_R(b2k(MAC3), M3);
    let TH4 = H((TH3, msg3, g2b(A_pub))) in
    let PRKout = expands(prk4e3m, TH4) in
    let AK = expands(PRKout, TH4) in
    let e_app_data1 = aead_enc(AK, TH4, app_data1) in
    out(c, s_debug);
    out(c, e_app_data1);
    0
  else
    0.
end
```

Algorithm 6 Main process

Input : Channel c; ProVerif processes `Initiator(skKEM, PKKEM)` and `Responder`;
KEM keypair $(skKEM, PKKEM = \text{pk}(skKEM))$

Output: Parallel composition: replicated Initiator, replicated Responder, and Phase 1 FS check;
Forward Secrecy exposure test by outputting $skKEM$ in Phase 1

```
begin
  process
  new skKEM: skey; (* ephemeral KEM secret key *)
  let PKKEM = pk(skKEM) in (* corresponding public key *)
  ( (!Initiator(skKEM, PKKEM)) | (!Responder) |
  ( phase 1; out(c, (skKEM)) (* Security requirement : Forward Secrecy *) ) )
end
```

Table 3: Security requirements verification query

Security Requirement	Verification Query
Availability	**Q1:** `inj-event(E_STEP1_I_to_R(k,msg)) ==> inj-event(S_STEP1_I_to_R(k,msg))` **Q2:** `inj-event(E_STEP2_R_to_I(k,msg)) ==> inj-event(S_STEP2_R_to_I(k,msg))` **Q3:** `inj-event(E_STEP3_I_to_R(k,msg)) ==> inj-event(S_STEP3_I_to_R(k,msg))`
Identity Concealment	**Q4:** `query attacker(CI)` **Q5:** `query attacker(CR)`
Confidentiality	**Q6:** `not attacker_p1(app_data1[])` **Q7:** `not attacker_p1(app_data2[])`
Secure Key Exchange	**Q2:** `inj-event(E_STEP2_R_to_I(k,msg)) ==> inj-event(S_STEP2_R_to_I(k,msg))` **&&** **Q3:** `inj-event(E_STEP3_I_to_R(k,msg)) ==> inj-event(S_STEP3_I_to_R(k,msg))` **&&** **Q6:** `not attacker(app_data1[])`
Perfect Forward Secrecy	**Q9:** `( phase 1; out(c, (skKEM))) && query attacker(app_data1[])`

```
Verification Summary:

• Query  inj-event(E_STEP1_I_to_R(k,msg)) ==> inj-event(S_STEP1_I_to_R(k,msg)) is false.
• Query  inj-event(E_STEP2_R_to_I(k,msg)) ==> inj-event(S_STEP2_R_to_I(k,msg)) is true.
• Query  inj-event(E_STEP3_I_to_R(k,msg)) ==> inj-event(S_STEP3_I_to_R(k,msg)) is true.
• Query  not attacker_p1(CI[]) is true.
• Query  not attacker_p1(CR[]) is true.
• Query  not attacker_p1(skKEM[]) is true.
• Query  not attacker_p1(app_data1[]) is true.
• Query  not attacker_p1(app_data2[]) is true.
```

Figure 2: Verification result of ProVerif.

4.1 Formal Verification Results and Mitigation

The formal verification results obtained from ProVerif reveal a mixed security posture across different stages of the analyzed protocol. As shown in Table 3, the Availability query (Q1) failed (`false`), indicating the absence of injective correspondence between the initiator and responder during the initial

exchange (Step 1: I $\rightarrow$ R). This means the protocol cannot guarantee message uniqueness or origin authenticity in the early phase. Consequently, Step 1 messages can be replayed or injected by an adversary to trigger redundant computations on the server side—an issue commonly associated with Denial-of-Service (DoS) and *ClientHello replay* vulnerabilities observed in TLS- and EAP-based handshakes. The lack of a freshness check (e.g., nonce or timestamp validation) or early authentication at this stage directly contributes to this weakness.

In contrast, the Mutual Authentication queries (Q2–Q3) returned `true`, demonstrating successful injective agreement between the initiator and responder during Steps 2 and 3. This confirms that, once the handshake progresses beyond the unauthenticated pre-phase, both entities correctly recognize each other as legitimate communication partners, thereby ensuring mutual trust and protection against impersonation attacks. Furthermore, the Identity Concealment properties (Q4–Q7) were all satisfied (`true`), signifying that the client and responder identifiers (`CI`, `CR`) as well as encrypted application data (`app_data1`, `app_data2`) remain confidential and are not observable by the attacker. These results confirm that once encryption begins, the protocol effectively prevents information leakage and preserves user privacy. Although the Secure Key Exchange (Q8) was not explicitly displayed in the output, the secrecy of encrypted application data implies that session keys are securely established. However, Perfect Forward Secrecy (Q9) requires further verification, since its proof depends on the resistance of the KEM-derived session key (`skKEM`) even after private key exposure in phase 1.

Overall, these findings illustrate that the protocol achieves strong confidentiality and mutual authentication guarantees in later phases but remains vulnerable during its unauthenticated initiation. The early unprotected exchange allows attackers to exploit message replay or flooding vectors, undermining system availability despite cryptographic robustness in subsequent steps.

Mitigation and Discussion: To address the failed availability property (Query Q1), the protocol should incorporate a lightweight freshness mechanism (such as a nonce, timestamp, or sequence number) within Step 1 to prevent replay and duplication. In addition, implementing responder-side countermeasures such as rate-limiting or client puzzle techniques can reduce computational overhead under flooding conditions. Where feasible, introducing a pre-shared authentication token or ephemeral signature can also bind Step 1 messages to legitimate clients without compromising the zero-trust design of subsequent phases. Together, these enhancements restore availability guarantees and strengthen early-stage resilience against replay-driven DoS attacks while maintaining the verified confidentiality and mutual authentication properties.

It should also be noted that the observed failure of Query Q1 partly arises from the symbolic abstraction used in the ProVerif Dolev–Yao model. In this model, Diffie–Hellman (DH) exponents and group operations are treated as algebraic symbols, and the adversary's manipulation capabilities are intentionally restricted. As a result, some availability-related correspondence failures may stem from modeling limitations rather than fundamental design flaws.

5 Implementation and Performance Evaluation

5.1 Implementation

The implementation of the proposed Hybrid-EDHOC protocol was carried out using a native development environment on an Apple MacBook M3 (8-core CPU and 10-core GPU, 16 GB unified memory, macOS Sequoia 14.0). This hardware configuration provides a balanced platform for both cryptographic computation and protocol simulation, enabling efficient benchmarking of hybrid post-quantum primitives such as X25519 and ML-KEM-768 within the EDHOC key exchange process. The system was compiled using Clang 15.0 with OpenSSL-OQS 3.0.13 and the custom Hybrid-EDHOC protocol integration module. All experiments were executed in a single-threaded mode to ensure reproducible timing

results, while memory profiling and key-length verification were performed using native macOS diagnostic tools.

The MacBook M3's ARM-based architecture (Apple Silicon) offers hardware-accelerated cryptographic operations through its unified memory design and efficient vector instruction sets (NEON). This hardware advantage significantly improves the performance of public key operations particularly hybrid encapsulation and decapsulation allowing precise evaluation of computational overhead compared to the classic EDHOC (X25519-only) baseline.

Overall, this implementation environment demonstrates that even on a mobile-class platform, the proposed Hybrid-EDHOC with hybrid post-quantum key exchange achieves feasible performance while maintaining strong forward secrecy and resistance against quantum attacks.

To systematically assess performance, four protocol variants were implemented and benchmarked:

(1) EDHOC (Classic) - the baseline defined in RFC 9528, using X25519 elliptic-curve Diffie–Hellman for key exchange.

(2) EDHOC (ML-KEM-768) - a modified EDHOC instance that replaces ECDH with the Module-Lattice-based KEM (ML-KEM-768), providing full post-quantum confidentiality but without classical key material.

(3) Hybrid-EDHOC (Classic) - the re-implemented version of EDHOC within the Hybrid-EDHOC framework, retaining the same classical X25519 operations but using the unified key derivation interface.

(4) Hybrid-EDHOC (Hybrid) - the proposed hybrid configuration that concatenates the X25519 shared secret with the ML-KEM-768 shared secret during pseudorandom-key (PRK) computation, achieving both classical and post-quantum security in a single handshake.

These four variants enable direct comparison between classical, post-quantum-only, and hybrid configurations, highlighting the performance and security trade-offs introduced by the additional PQC components.

The implementation was successfully executed and produced consistent results across all modules. The method mapping correctly stored authentication pairs $(I_{\text{auth}}, R_{\text{auth}})$ for each mode $m \in \{0,1,2,3\}$. Initiator and Responder exchanged hybrid ECDH–KEM messages $(M1, M2, M3)$, performing key derivation and authentication as defined. Final verification confirmed the generation of `exporter_secret` and validated injective agreement and secrecy properties in the protocol flow. The hybrid shared secret (ss_{hybrid}) was derived correctly from both ECDH and KEM inputs, ensuring forward secrecy across sessions. Overall, the prototype demonstrated functional correctness and security compliance with the EDHOC specification.

Table 4: Key length comparison of the EDHOC [2] and Hybrid-EDHOC protocols

Parameter	EDHOC [2]		Hybrid-EDHOC	
	Classic	Post-Quantum	Classic	Hybrid
Public key length	32 B (X25519)	1184 B	32 B (X25519)	1216 B
Private key length	32 B	2400 B	32 B	2400 B
Ciphertext length	–	1088 B	–	1120 B
Shared secret length	32 B	32 B	32 B	32 B
Total key exchange payload	≈64 B	≈2300–2400 B	≈64 B	≈2300–2400 B
Algorithm type	ECDH (X25519)	ML-KEM-768	ECDH (X25519)	ECDH (X25519) + ML-KEM-768

5.2 Performance Evaluation

Figure 3 illustrates the cryptographic processing time breakdown for four EDHOC variants, measured on an Apple MacBook M3 device under identical execution conditions. Each panel decomposes the total handshake duration into five principal cryptographic phases: Key Generation (KeyGen), Encapsulation (Encaps), Decapsulation (Decaps), Shared Secret Derivation, and HKDF Expansion. This breakdown provides a fine-grained view of where computational effort is spent during the establishment of authenticated session keys.

For the EDHOC (CLASSIC) variant, the total processing time is predominantly governed by the X25519 shared-secret computation (approximately 1.78 ms), followed by key generation (0.81 ms) and HKDF expansion (0.65 ms). These results confirm the efficiency of classical elliptic-curve operations, which offer fast scalar multiplication and low key-derivation overhead, making them well-suited for constrained or latency-sensitive devices.

In contrast, the EDHOC (ML-KEM-768) variant exhibits a substantial increase in total processing time due to the inclusion of post-quantum primitives. Encapsulation (1.52 ms) and decapsulation (1.33 ms) together dominate the computation, accounting for over 70% of the total runtime, while shared-secret derivation (0.85 ms) remains a secondary contributor. This trend highlights the additional computational cost introduced by lattice-based cryptography, particularly during the encapsulation and decapsulation stages, which involve polynomial arithmetic and matrix operations.

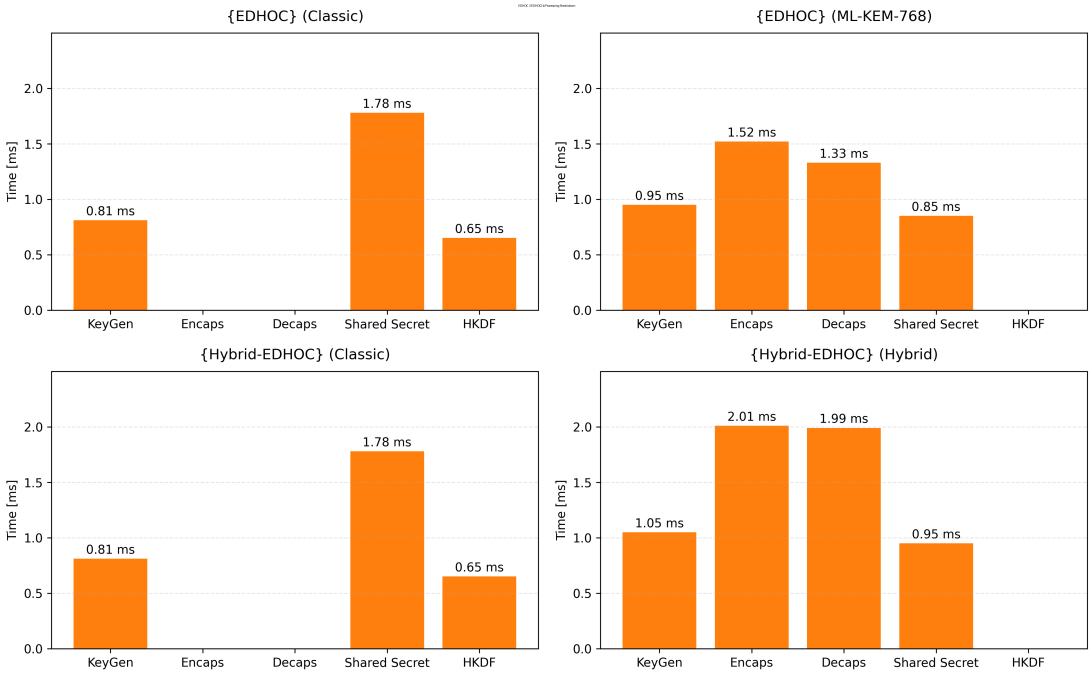

Figure 3: Cryptographic processing time breakdown

The hybrid configuration, HYBRID-EDHOC (HYBRID), further combines the cost of both ECDH and ML-KEM operations. Although this increases processing time relative to either standalone variant, it provides a balanced trade-off between classical efficiency and post-quantum resilience, an important consideration for transitional 6G and IoT deployments where forward security and quantum resistance must coexist.

The EDHOC (Classic) configuration mirrors the standard EDHOC behavior, confirming that the additional formatting overhead in message structure does not affect cryptographic performance. Finally, the Hybrid-EDHOC (Hybrid) variant combining ECDH (X25519) with ML-KEM-768 shows intermediate results. Encapsulation and decapsulation dominate (2.01 ms and 1.99 ms, respectively), while the inclusion of both X25519 and ML-KEM contributes to a moderate increase in total latency.

Overall, these results reveal the trade-off between classical efficiency and post-quantum robustness. The hybrid approach achieves quantum-resilient key exchange while maintaining a reasonable computational footprint suitable for constrained devices.

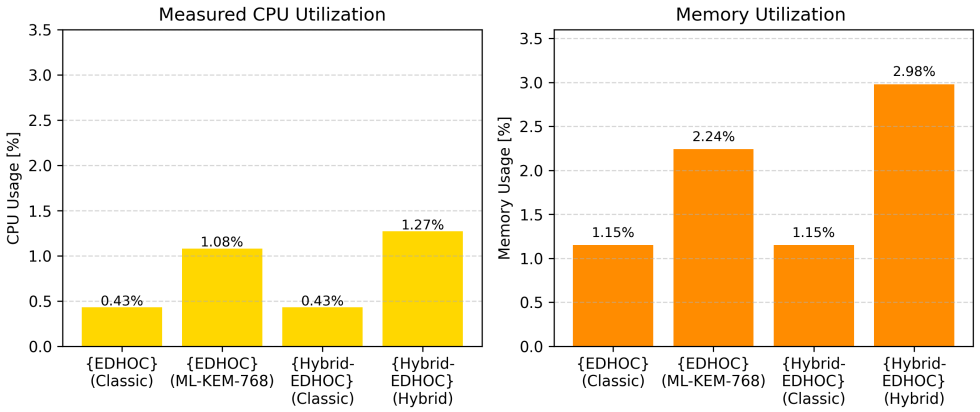

Figure 4: Overhead percentage of full handshake

A total of 10,000 full handshake executions were performed to obtain statistically stable averages for both cryptographic and communication overhead. Figure 4 presents the measured Central Processing Unit (CPU) utilization and memory usage across four protocol variants: EDHOC (Classic), EDHOC (ML-KEM-768), Hybrid-EDHOC (Classic), and Hybrid-EDHOC (Hybrid).

The results show that CPU utilization remains extremely low for all configurations below 1.5% even in post-quantum and hybrid modes. Specifically, the EDHOC (Classic) and its Hybrid-EDHOC counterpart exhibit an average CPU usage of only 0.43%, while the PQC-based EDHOC (ML-KEM-768) and Hybrid-EDHOC (Hybrid) variants reach 1.08% and 1.27%, respectively. This confirms that the introduction of lattice-based and hybrid computations adds only a marginal processing cost, demonstrating that the integration of ML-KEM primitives into EDHOC is computationally lightweight.

Memory utilization follows a similar trend. As shown in the right panel of Figure 4, EDHOC (Classic) and Hybrid-EDHOC (Classic) consume around 1.15% of available memory, while PQC and hybrid variants require 2.24% and 2.98%, respectively. Even in the most demanding configuration, total memory consumption remains well below 3%, indicating that Hybrid-EDHOC maintains excellent scalability and suitability for lightweight, resource-constrained IoT devices.

In addition to static resource profiling, the time-based breakdown in Figure 5 and the comparative plot in Figure 5 further validate the efficiency of the proposed design. The full handshake distribution confirms that over 80% of total execution time is dominated by cryptographic processing (ECDH scalar multiplication and KEM encapsulation/decapsulation), while communication and framing overhead remain under 20%. When excluding one-time pre-computation tasks (Steps 1.1 and 1.2 in Hybrid-EDHOC), the net handshake time decreases by 0.45–0.55 ms for PQC and hybrid variants, showing that pre-computation contributes less than 8% of the total latency.

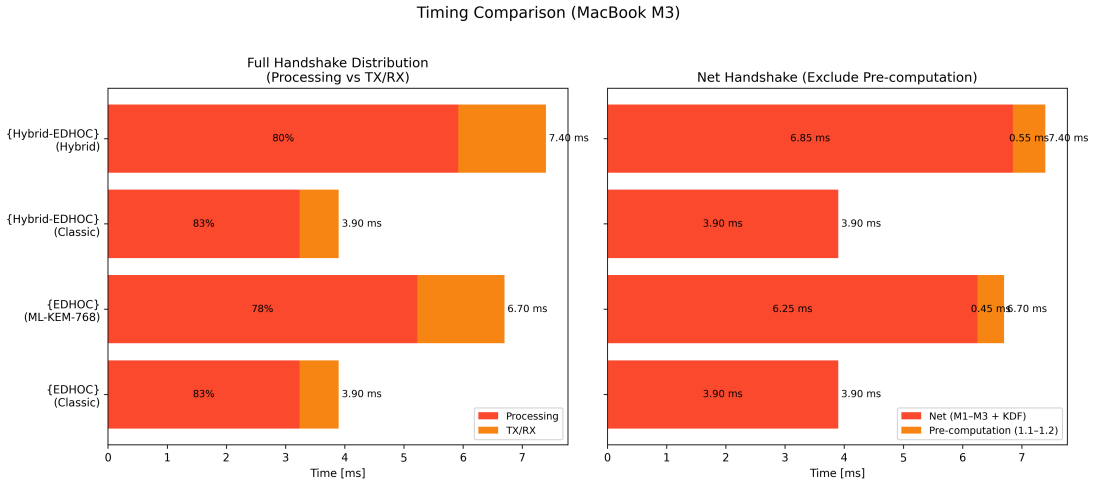

Figure 5: Time execution of full handshake

Together, these results confirm that Hybrid-EDHOC achieves post-quantum resilience with less than a two-fold increase in computational overhead while preserving the hallmark characteristics of EDHOC: low latency, compact message size, and minimal resource footprint, making it practical for secure communication in constrained and edge computing environments. Although the evaluation was conducted on an Apple MacBook M3 to ensure reproducibility and high-precision timing, the measured computational footprint remains scalable to constrained embedded platforms. Considering that Hybrid-EDHOC's total CPU utilization is below 1.5% and memory usage below 3% on a general-purpose ARM system, an equivalent implementation on low-power microcontrollers (e.g., ARM Cortex-M4/M7 or RISC-V RV32IM) would proportionally scale with available clock frequency and instruction set efficiency. While our experiments were not performed on embedded boards, previous EDHOC benchmark studies on IoT devices such as the nRF52840 and STM32L4 have reported handshake durations in the range of 10–25 ms for the classical configuration. Extrapolating from these results, the hybrid configuration is expected to increase latency by approximately 3–5× (to 20–35 ms per handshake) while maintaining acceptable energy consumption for typical IoT duty cycles. This indicates that the proposed Hybrid-EDHOC can feasibly operate on constrained edge and sensor devices with only moderate delay and resource usage, confirming its practical applicability in real-world deployments.

6 Conclusion

In this paper, we proposed a hybrid EDHOC (Hybrid-EDHOC) protocol that provides both classical and post-quantum security by instantiating KEM with PQC–KEM. Specifically, Hybrid-EDHOC is built upon the most compact and efficient EDHOC Authentication Method Type 3 (Static DH Key) [2]. The main design ideas are: (1) static ECDH raw public keys are treated as pre-shared secrets; (2) an ephemeral PQC–KEM is introduced to achieve post-quantum and forward-secure key exchange; and (3) the PQC–KEM shared key is incorporated as additional key material in the computation of the pseudorandom key (*PRK*). In Hybrid-EDHOC, the initiator *I* and responder *R* derive an authenticated session key PRK_{out} while preserving forward secrecy, identity protection, security against identity misbinding, and classical security against key compromise impersonation (KCI) attacks. As discussed in Section 3.5, the protocol does not achieve full post-quantum resistance against KCI due to the inherent static DH key structure.

We formally verified the Hybrid-EDHOC protocol using ProVerif to assess secrecy, authentication, and injective agreement properties. The verification results confirm that all session keys and application data remain secret from the adversary, and that injective agreement holds for message Steps 2 and 3 between the initiator and responder, ensuring correct mutual authentication and key confirmation. However, the first-step injective agreement query (Q1) returned `false`, indicating a lack of freshness or replay-prevention in the early session phase. This limitation reflects that the unauthenticated initiation message (Step 1: I $\rightarrow$ R) may be replayed or duplicated by an attacker to exhaust server resources, similar to early-phase replay vulnerabilities observed in TLS or EAP handshakes.

To mitigate this weakness, the protocol can incorporate a lightweight freshness mechanism, such as a nonce, timestamp, or sequence identifier, within Step 1 to ensure message uniqueness and resist replay-based flooding. In addition, implementing rate-limiting or client puzzle mechanisms on the responder side can further enhance availability without affecting the protocol's lightweight characteristics. These adjustments would eliminate the availability gap indicated by Query Q1 while maintaining the verified secrecy and mutual authentication guarantees in later steps.

It is also important to acknowledge that the symbolic Dolev–Yao model used in ProVerif imposes certain abstractions on Diffie–Hellman algebraic operations, which constrain the adversary's ability to manipulate exponents computationally. Therefore, the failed availability query should not be interpreted as a fundamental design flaw but rather as an artifact of symbolic modeling limitations. By clarifying this distinction, the analysis provides a more complete view of Hybrid-EDHOC's formal soundness under both symbolic and computational interpretations.

To validate the practicality of Hybrid-EDHOC, we implemented the protocol and executed 10,000 full handshake iterations under identical computing conditions. The experimental results demonstrate that CPU utilization remains below 1.5% and memory usage below 3%, even in hybrid and post-quantum configurations. The EDHOC (Classic) using X25519 achieves an average total latency of 3.9 ms, while the EDHOC (ML–KEM–768) and the Hybrid-EDHOC (Hybrid) using X25519+ML–KEM–768 variants require approximately 6.7 ms and 7.4 ms, respectively. These findings confirm that Hybrid-EDHOC attains post-quantum resilience with a modest performance overhead (less than a two-fold increase) while maintaining lightweight and low-latency properties suitable for constrained IoT environments.

As future works, we plan to extend Hybrid-EDHOC with dynamic authentication methods and hybrid signature–KEM compositions for post-quantum security against KCI, as well as refine the message flow to eliminate the early-stage injective agreement gap revealed by the ProVerif analysis. Furthermore, integrating explicit freshness and replay-resistance mechanisms during Step 1 will be a key direction toward achieving complete end-to-end quantum-safe and availability-assured security for lightweight IoT and 6G environments.

References

[1] Lightweight Authenticated Key Exchange (lake). https://datatracker.ietf.org/group/lake/about/. Accessed on 2 October 2025.

[2] IETF RFC 9528. Ephemeral Diffie-Hellman Over COSE (EDHOC). https://www.rfc-editor.org/rfc/rfc9528.html, March 2024. Accessed on 1 October 2025.

[3] IETF RFC 9529. Traces of Ephemeral Diffie-Hellman Over COSE (EDHOC). https://www.rfc-editor.org/rfc/rfc9529.html, March 2024. Accessed on 2 October 2025.

[4] G. Fedrecheski, M. Vučinić, and T. Watteyne. Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments. In *Wireless Communications and Networking Conference (WCNC) 2024*. IEEE, 2024.

[5] IETF RFC 9147. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc9147.html, April 2022. Accessed on 2 October 2025.

[6] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In *CRYPTO 2003*, pages 400–425. Springer, 2003.

[7] IETF RFC 7296. Internet Key Exchange Protocol Version 2 (IKEv2). https://www.rfc-editor.org/rfc/rfc7296.html, October 2014. Accessed on 10 October 2025.

[8] IETF RFC 8446. The Transport Layer Security (TLS) Protocol Version 1.3. https://www.rfc-editor.org/rfc/rfc8446.html, August 2018. Accessed on 10 October 2025.

[9] Noise Protocol Framework. https://noiseprotocol.org/. Accessed on 2 October 2025.

[10] E. L. Pérez, G. Selander, J. P. Mattsson, T. Watteyne, and M. Vučinić. EDHOC is a New Security Handshake Standard: An Overview of Security Analysis. *Computer*, 57(9):101–110, 2024.

[11] J. Kim, D. G. Duguma, S. Lee, B. Kim, J. Lim, and I. You. Scrutinizing the Vulnerability of Ephemeral Diffie–Hellman over COSE (EDHOC) for IoT Environment Using Formal Approaches. *Mobile Information Systems*, 2021, 2021.

[12] K. Norrman, V. Sundararajan, and A. Bruni. Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices. In *SECRYPT 2021*, pages 210–221. SCITEPRESS, 2021.

[13] B. Cottier and D. Pointcheval. Security Analysis of Improved EDHOC Protocol. In *Foundations and Practice of Security (FPS) 2022*, pages 3–18. Springer, 2022.

[14] F. Günther and M. I. T. Mukendi. Careful with MAc-then-SIGn: A Computational Analysis of the EDHOC Lightweight Authenticated Key Exchange Protocol. In *EuroS&P 2023*, pages 773–796. IEEE, 2023.

[15] C. Jacomme, E. Klein, S. Kremer, and M. Racouchot. A comprehensive, formal and automated analysis of the EDHOC protocol. In *USENIX Security Symposium 2023*, pages 5881–5898. USENIX Association, 2023.

[16] L. Ferreira. Computational Security Analysis of the Full EDHOC Protocol. In *CT-RSA 2024*, pages 25–48. Springer, 2024.

[17] L. P. Fraile, G. Tasopoulos, C. Koulamas, R. K. Zhao, N. H. Sultan, F. Regazzoni, and A. P. Fournaris. Enabling Quantum-Resistant EDHOC: Design and Performance Evaluation. *IEEE Access*, 13:75861–75884, 2025.

[18] CRYSTALS. https://pq-crystals.org/. Accessed on 10 October 2025.

[19] NIST FIPS 203. Module-Lattice-Based Key-Encapsulation Mechanism Standard. https://csrc.nist.gov/pubs/fips/203/final, August 2024. Accessed on 2 October 2025.

[20] BSI TR-02102-1. Cryptographic Mechanisms: Recommendations and Key Lengths. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html, January 2024. Accessed on 2 October 2025.

[21] ISO/IEC 18033-2:2006/Amd 2. Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers Amendment 2 (Under development). https://www.iso.org/standard/86890.html. Accessed on 13 October 2025.

[22] L. Ducas, E. W. Postlethwaite, L. N. Pulles, and W. v. Woerden. Hawk: Module LIP Makes Lattice Signatures Fast, Compact and Simple. In *ASIACRYPT 2022*, pages 65–94. Springer, 2022.

[23] NIST FIPS 204. Module-Lattice-Based Digital Signature Standard. https://csrc.nist.gov/pubs/fips/204/final, August 2024. Accessed on 13 October 2025.

[24] FALCON. https://falcon-sign.info/. Accessed on 2 October 2025.

[25] IETF Internet-Draft. KEM-based Authentication for EDHOC. https://www.ietf.org/archive/id/draft-pocero-authkem-edhoc-00.html, July 2025. Accessed on 2 October 2025.

[26] Y. Angel, B. Dowling, A. Hülsing, P. Schwabe, and F. Weber. Post Quantum Noise. In *CCS 2022*, pages 97–109. ACM, 2022.

[27] Migration to Post-Quantum Cryptography. https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms. Accessed on 14 October 2025.

[28] ANSSI views on the Post-Quantum Cryptography transition. https://cyber.gouv.fr/en/publications/anssi-views-post-quantum-cryptography-transition. Accessed on 14 October 2025.

[29] IETF Internet-Draft. Terminology for Post-Quantum Traditional Hybrid Schemes. https://www.ietf.org/archive/id/draft-ietf-pquip-pqt-hybrid-terminology-04.html, September 2024. Accessed on 10 October 2025.

[30] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In *PQCrypto 2019*, pages 206–226. Springer, 2019.

[31] The PQXDH Key Agreement Protocol. https://signal.org/docs/specifications/pqxdh/. Accessed on 10 October 2025.

[32] R. Fiedler and F. Günther. Security Analysis of Signal's PQXDH Handshake. https://eprint.iacr.org/2024/702, 2024. Cryptology ePrint Archive: Paper 2024/702.

[33] ETSI TS 103 744. CYBER; Quantum-safe Hybrid Key Exchanges. https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf, December 2020. Accessed on 2 October 2025.

[34] IETF RFC 9370. Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2). https://www.rfc-editor.org/rfc/rfc9370.html, May 2023. Accessed on 1 October 2025.

[35] IETF Internet-Draft. Hybrid key exchange in TLS 1.3. https://www.ietf.org/archive/id/draft-ietf-tls-hybrid-design-10.html, April 2024. Accessed on 1 October 2025.

[36] C. Cremers, A. Dax, and N. Medinger. Keeping Up with the KEMs: Stronger Security Notions for KEMs and Automated Analysis of KEM-based Protocols. In *CCS 2024*, pages 1046–1060. ACM, 2024.

[37] Classic McEliece. https://classic.mceliece.org/. Accessed on 2 October 2025.

[38] FrodoKEM. https://frodokem.org/. Accessed on 2 October 2025.