Symbolic forward secrecy analysis of ML-KEM

[1*]Varsha Jarali and [1]Shashi Kant Pandey

[1]Dayananda Sagar University, Bengaluru, India, 562112,
[2]Society for Electronic Transactions and Security (SETS),
(Under the Office of the Principal Scientific Adviser to the Government of India)
Taramani, Chennai-600113, India.
varshapjarali@gmail.com
shashikantshvet@gmail.com

Abstract. Post-quantum Key Encapsulation Mechanisms (KEMs) rely on generic transformations to achieve active security. Specifically, ML-KEM (FIPS 203) applies the Fujisaki–Okamoto (FO) transform to a lattice-based public-key encryption scheme, Kyber.CPAPKE, to construct an IND-CCA2 secure key exchange protocol, ML-KEM. While NIST standards and associated special publications discuss guidelines for establishing larger shared keying material using ML-KEM, formal symbolic proofs to evaluate these operational setups remain sparse. This work addresses this gap by conducting a formal symbolic analysis of ML-KEM using ProVerif. First, we verify the core primitive, establishing symbolic proofs for session key confidentiality, key agreement soundness, and Fujisaki–Okamoto implicit rejection fallback secrecy. Second, we examine the temporal security guarantees of ML-KEM under long-term static key leakage. Using ProVerif's global phase constructs, we demonstrate that standalone static ML-KEM does not preserve Perfect Forward Secrecy (PFS), rendering repeated static encapsulation for additional key share establishment vulnerable to retroactive decapsulation. Lastly, we formally model and verify that deploying ML-KEM with ephemeral key rotation or ratcheting mechanisms mitigates this vulnerability, providing robust forward secrecy and complete session domain separation.

Keywords: MLKEM · Formal verification · ProVerif · Post Quantum Cryptography · NIST
Mathematics Subject Classification (MSC2020) 133574 · 03D05 . 03B38

1 Introduction

A growing field of research, sometimes called computer-aided cryptography, is concerned with the formal analysis of the design and implementation of cryptographic mechanisms and protocols. Many of the tools and techniques developed in this field can be used to verify protocol components. Domain-specific protocol verification tools have been developed to analyse the correctness and security of cryptographic protocols and modules [1], [2], [3], [4], [5], [6], [7]. Another line

of work has focused on developing formally verified cryptographic libraries in C and assembly. Furthermore, the literature shows that different tools are better at different verification tasks. In particular, targeted security-oriented tools are better at analysing cryptographic components, while standard software verification tools are effective on the rest. Combining these tools to verify a full protocol implementation remains a challenge. Recent work [8] addressed formal verification of practical cryptographic protocol implementations written in Rust and showed how to develop machine-checked proofs of diverse properties such as runtime safety, parsing correctness, and cryptographic protocol security. [9] presented a formally verified machine-checked proof of the correctness and IND-CCA security of ML-KEM in EasyCrypt [10]. Using the automated theorem prover Isabelle [3], the author [11], formalised the algorithm specifications and parameter sets of Kyber's public-key encryption scheme and verified the $\delta-$ correctness and indistinguishability under the chosen-plaintext attack property. A recent update is available in [12] for verification of IND-CPA security proof and properties for CRYSTALS-KYBER using the CryptHOL library[13].

In this paper, we formally verify the ML-KEM protocol in the ProVerif tool.

2 Results and Analysis of ML-KEM

As specified in the NIST ML-KEM standard (FIPS 203), we modelled the IND-CCA2-secure key generation (`KeyGen`), encapsulation (`Encaps`), and decapsulation (`Decaps`) mechanisms for symbolic formal verification in ProVerif. The `KeyGen` algorithm generates the underlying CPA key pair $(pk_\text{cpa}, sk_\text{cpa})$, computes $h_pk = H(pk_\text{cpa})$, and embeds a 32-byte fallback seed z to construct the full CCA secret key $sk = (sk_\text{cpa} \parallel pk_\text{cpa} \parallel h_pk \parallel z)$. During `Encaps`, an ephemeral seed m is sampled to derive $(\bar{K}, r) = G(m \parallel H(pk))$, producing the ciphertext $c = \text{PKE.Enc}(pk, m, r)$ and the shared key $K = \text{KDF}(\bar{K} \parallel H(c))$. Finally, `Decaps` recovers $m' = \text{PKE.Dec}(sk_\text{cpa}, c)$ and re-encrypts $c' = \text{PKE.Enc}(pk, m', r')$; under the Fujisaki–Okamoto (FO) transform, matching ciphertexts ($c = c'$) yield the shared key $K = \text{KDF}(\bar{K}' \parallel H(c))$, whereas a mismatch triggers implicit rejection to return $K_\text{fail} = \text{KDF}(z \parallel H(c))$.

2.1 Security Model

We formally verified ML-KEM across three distinct configuration approaches using ProVerif. Initially, modelling the standard NIST ML-KEM specification, we evaluated session key confidentiality (through encrypted secret payloads), implicit rejection secrecy (confirming that Fujisaki–Okamoto fallback keys derived under mismatching ciphertexts leak no private state), and key agreement soundness (proving that an honest Initiator and Responder exchanging identical ciphertexts c always establish matching session keys $k_1 = k_2$). Figure 1 gives a ProVerif summary of these security properties, which proves that ML-KEM maintains confidentiality and key agreement soundness. As a standalone ML-KEM is an unauthenticated public-key primitive, it does not inherently guaran-

tee sender authentication; authentication must be provided by embedding ML-KEM into higher-level transport protocols such as TLS 1.3 or Signal.

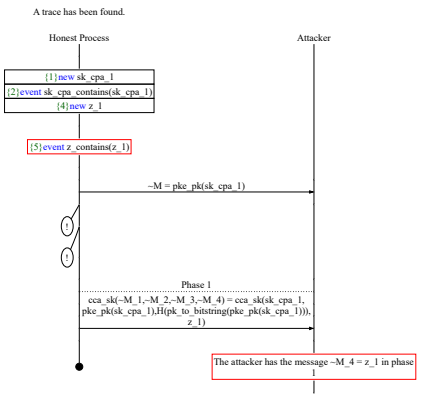
```
Verification summary:

Query not attacker(Secret_Payload[]) is true.

Query not attacker(Fail_Payload[]) is true.

Query event(initiator_encap(pk_1,c_1,k1)) && event(responder_decap(pk_1,c_1,k2)) ==> k1 = k2 is true.
```

Fig. 1: ProVerif output of security queries for ML-KEM for standard properties

Modeling Forward Secrecy via ProVerif Phase To formally evaluate Perfect Forward Secrecy (PFS), we leverage ProVerif's global synchronization construct (`Phase`). In Phase 0, honest participants execute the ML-KEM handshake, deriving a session key K to encrypt a test payload (`Secret_Payload_Phase0`) broadcast over a public channel. In Phase 1, long-term key compromise is simulated by leaking the static secret key (`out(c_pub, sk)`). The query `query attacker(Secret_Payload_Phase0)` evaluates to `false` (confirming the secret is reachable by the adversary), formally proving that standalone static ML-KEM lacks PFS against retroactive decapsulation. Conversely, incorporating fresh per-session ephemeral keys or key rotation isolates past session keys and successfully preserves PFS. Figure 2a lists the corresponding ProVerif queries.

```
(* --- Private Fail Payload for FO Rejection Secrecy --- *)
free Fail_Payload: bitstring [private].

(* 1. Secrecy of FO Fallback Seed z *)
query secret z.

(* 2. Secrecy of Per-Session Seed m  *)
query secret m.

(* 3. Secrecy of Pre-Key K_bar *)
query secret K_bar.

(* 4. Secrecy of Final Session Key K *)
query secret K.

(* 5. Secrecy of CPA Secret Key sk_cpa *)
query secret sk_cpa.

(* 6. Secrecy of FO Implicit Rejection Fail Payload *)
query attacker(Fail_Payload).

(* 7. Key Agreement / Soundness Query *)
query pk: pkey_cpa, c: ciphertext, k1: key, k2: key;
    event(initiator_encap(pk, c, k1)) && event(responder_decap(pk, c, k2)) ==> k1 = k2.
```

(a) Verification of perfect forward secrecy of ML-KEM



(b) Attack trace of reachability of past messages in PFS scenario

Fig. 2: queries in the ProVerif model and attack trace of ML-KEM

In the figure 3 below, ProVerif outputs `Query not attacker_p1(Fail_Payload[]) is false.` This query result directly reflects how Phase 1 static key compromise impacts the Fujisaki–Okamoto (FO) implicit rejection mechanism in ML-KEM. In ML-KEM (Kyber), when decapsulation receives an invalid or tampered ciphertext c, the Fujisaki–Okamoto (FO) transform triggers the implicit rejection path, deriving a fallback key:

$$K_\text{fail} = \text{KDF}(z \parallel H(c))$$

The long-term CCA secret key tuple is structured as $sk = (sk_\text{cpa} \parallel pk \parallel H(pk) \parallel z)$, where z is the static 32-byte fallback seed. In Phase 1, leaking sk directly exposes z to the attacker. With z and the recorded Phase 0 ciphertext c in hand, the attacker computes K_fail and decrypts the fallback payload `Fail_Payload`.

```
Verification summary:

Query secret z is false.

Query secret m is true.

Query secret K_bar is false.

Query secret K is false.

Query secret sk_cpa is false.

Query not attacker_p1(Fail_Payload[]) is false.

Query event(initiator_encap(pk_1,c_1,k1)) && event(responder_decap(pk_1,c_1,k2)) ==> k1 = k2 is true.
```

Fig. 3: ProVerif output on perfect forward secrecy of ML-KEM

The presented ProVerif summary is useful for generating an attack trace in the case of PFS. Here we present the trace of this attack in Figure 2b. It gives the attack trace of the reachability of the message (payload) of Phase 0.

Phase-Based Key Rotation and Session Isolation To evaluate the security impact of periodic key rotation, we implemented a multi-phase ProVerif model simulating key state transitions across execution phases. In Phase 0, the primary long-term key pair (sk_1, pk_1) and an ephemeral session key (K_1) are generated to encrypt a secret payload (`Secret_Payload1`). Upon transitioning to Phase 1, a key rotation occurs: fresh long-term keys (sk_2, pk_2) and new session keys (K_2) are generated to protect subsequent communications (`Secret_Payload2`). Simultaneously in Phase 1, we simulate long-term key exposure by leaking the Phase 0 secret key (sk_1) onto the public channel (`phase 1; out(c_pub, sk1)`). Verification reveals that while the adversary successfully derives `Secret_Payload1` retroactively using sk_1, the Phase 1 payload (`Secret_Payload2`) remains completely confidential.

(`RESULT not attacker(Secret_Payload2[]) is true`).

This formally proves that key rotation enforces cryptographic domain separation, ensuring that compromising a past long-term key does not impact future sessions running under rotated keys.

Standard ML-KEM-1024 outputs a fixed 256-bit (32-byte) shared secret key. When an application requires a 512-bit symmetric key, executing ML-KEM encapsulation twice against the same static long-term public key is strongly discouraged due to key-reuse risks. Instead, best practices dictate generating a fresh key pair (or rotating the long-term key) for the second encapsulation session by reinitiating a different z, or expanding the single 256-bit shared secret using an approved Key Derivation Function (KDF) such as HKDF or SHAKE-256.

References

[1] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial*, Accessed: Aug. 6, 2026, 2026. [Online]. Available: `https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf`

[2] B. Blanchet, A. Fromherz, C. Jacomme, and B. Lipp, *Cryptoverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial*, Accessed: Aug. 6, 2026, 2026. [Online]. Available: `https://cryptoverif.inria.fr/manual.pdf`

[3] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/hol—a proof assistant for higher-order logic*, Accessed: Aug. 6, 2026, 2018. [Online]. Available: `https://isabelle.in.tum.de/doc/tutorial.pdf`

[4] S. Meier, C. Cremers, D. Basin, et al., *The tamarin prover*, Accessed: Aug. 6, 2026. [Online]. Available: `https://tamarin-prover.com/`

[5] T. Coquand, G. Huet, C. Paulin-Mohring, B. Barras, Y. Bertot, P. Castéran, et al., *Rocq prover*, Accessed: Aug. 6, 2026. [Online]. Available: `https://rocq-prover.org/`

[6] Mizar Project, *Mizar proof checker*, Accessed: Aug. 6, 2026. [Online]. Available: `http://mizar.org/`

[7] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed., ser. Lecture Notes in Computer Science, vol. 6841, Heidelberg: Springer, Aug. 2011, pp. 71–90.

[8] K. Bhargavan, L. L. Hansen, F. Kiefer, J. Schneider-Bensch, and B. Spitters, "Formal security and functional verification of cryptographic protocol implementations in rust," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, Association for Computing Machinery, 2025, pp. 2729–2743. DOI: `10.1145/3719027.3765213`

[9] J. B. Almeida et al., "Formally verifying kyber," in *Advances in Cryptology – CRYPTO 2024*, L. Reyzin and D. Stebila, Eds., ser. Lecture Notes in Computer Science, vol. 14921, Cham: Springer, 2024. DOI: `10.1007/978-3-031-68379-4-12` [Online]. Available: `https://doi.org/10.1007/978-3-031-68379-4-12`

[10] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, "Easycrypt: A tutorial," in *Foundations of Security Analysis and Design VII*, ser. Lecture Notes in Computer Science, vol. 8604, Cham:

Springer, 2014. DOI: 10.1007/978-3-319-10082-1_6 [Online]. Available: https://doi.org/10.1007/978-3-319-10082-1_6

[11] K. Kreuzer, *Verification of correctness and security properties for CRYSTALS-KYBER*, Cryptology ePrint Archive, Paper 2023/087, 2023. [Online]. Available: https://eprint.iacr.org/2023/087

[12] K. Kreuzer, "Verification of correctness and security properties for CRYSTALS-KYBER," in *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, IEEE, 2024, pp. 511–526. DOI: 10.1109/CSF61375.2024.00016

[13] A. Lochbihler, "Crypthol," *Archive of Formal Proofs*, May 2017, https://isa-afp.org/entries/CryptHOL.html, Formal proof development, ISSN: 2150-914x.