cryptography

MDPI

Article

A Post-Quantum End-to-End Secure Protocol for Instant Messaging Applications

Alfonso F. De Abiega-L'Eglisse [1], Kevin A. Delgado-Vargas [2], Humberto A. Ortega Alcocer [3], Gina Gallegos-García [2,*] and Eliseo Sarmiento-Rosales [1]

[1] Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional, Av. Instituto Politécnico Nacional s/n Edificio 9 Unidad Profesional Adolfo López Mateos, San Pedro Zacatenco, Nueva Industrial Vallejo, Gustavo A. Madero, Mexico City 07738, Mexico; adeabiegal1900@alumno.ipn.mx (A.F.D.A.-L.); esarmiento@ipn.mx (E.S.-R.)
[2] Centro de Investigación en Computación, Instituto Politécnico Nacional, Av. Juan de Dios Bátiz S/N, Nueva Industrial Vallejo, Gustavo A. Madero, Mexico City 07738, Mexico; kdelgadov2019@cic.ipn.mx
[3] Escuela Superior de Cómputo, Instituto Politécnico Nacional, Av. Juan de Dios Bátiz S/N, Nueva Industrial Vallejo, Gustavo A. Madero, Mexico City 07738, Mexico; hortegaa1500@alumno.ipn.mx
* Correspondence: ggallegosg@ipn.mx

Abstract

Modern instant messaging systems require end-to-end (E2E) security guarantees while operating over server-mediated infrastructures that cannot be fully trusted. At the same time, the impending transition to post-quantum cryptography raises nontrivial challenges for the design of secure messaging protocols that preserve these guarantees. In this work, we present the design of a post-quantum end-to-end secure protocol for instant messaging applications under an untrusted relay model. The proposed construction relies on lattice-based primitives standardized by NIST, namely ML-KEM for key establishment and ML-DSA for authentication, and follows a Double-KEM pattern combined with explicit context binding to derive an E2E session key known only to the communicating clients. The server acts solely as an authenticated relay and never gains access to plaintext messages or session keys. In addition to the protocol design, we complement the protocol description with an automated symbolic verification using ProVerif, establishing injective mutual authentication and session-key secrecy under a Dolev–Yao adversary model. Finally, we characterize the computational cost of different authentication and verification policies and evaluate the performance of the handshake on heterogeneous cloud-based architectures. The results provide practical insight into the feasibility of deploying post-quantum end-to-end secure protocols within existing instant messaging infrastructures.

Keywords: end-to-end secure messaging; post-quantum cryptography; secure protocol design; untrusted relay model; lattice-based cryptography; ML-KEM; ML-DSA

Check for updates

Academic Editor: Kris Gaj

Received: 25 January 2026
Revised: 10 March 2026
Accepted: 3 April 2026
Published: 23 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Instant messaging (IM) applications have become a fundamental component of modern digital communication, supporting personal, professional, and automated interactions over global network infrastructures. Due to the sensitive nature of the information exchanged, contemporary IM applications are expected to provide strong security guarantees, most notably confidentiality and authentication under an end-to-end (E2E) encryption model. In such applications, messages must remain confidential even in the presence

of compromised or curious servers that act solely as communication relays. This security paradigm has been widely adopted by modern messaging platforms and analyzed extensively in the literature [1–5].

The cryptographic foundations of most deployed E2E messaging protocols rely on number-theoretic assumptions such as integer factorization and the discrete logarithm problem, which underpin widely used primitives including RSA and elliptic curve cryptography (ECC) [6,7]. However, the advent of large-scale quantum computers poses a significant threat to these assumptions. Shor's algorithm demonstrates that both integer factorization and discrete logarithms can be solved in polynomial time on a quantum computer, rendering classical public-key primitives insecure in such a setting [8]. This realization has motivated a global transition toward post-quantum cryptography (PQC), culminating in the recent standardization efforts led by the National Institute of Standards and Technology (NIST), which selected lattice-based schemes such as ML-KEM (CRYSTALS-Kyber) and ML-DSA (CRYSTALS-Dilithium) as primary candidates for post-quantum key establishment and digital signatures, respectively [9–11]. While the integration of post-quantum primitives into traditional secure communication protocols has been actively studied, the design of post-quantum secure protocols for instant messaging applications presents unique challenges. Unlike transport-layer protocols such as TLS [12], IM applications must preserve strict E2E confidentiality guarantees while operating over server-mediated infrastructures that cannot be fully trusted. Simply replacing classical primitives with post-quantum counterparts in existing designs is insufficient to address these constraints and may introduce subtle vulnerabilities related to authentication, key agreement semantics, or context binding [13,14]. Prior analyses of IM protocols, including Signal and Telegram's MTProto, have highlighted how design choices at the protocol level critically impact security guarantees, even when strong cryptographic primitives are employed [1,3,14,15].

Several efforts have explored post-quantum secure messaging or adaptations of existing protocols to the post-quantum setting [16–18]. These works demonstrate the feasibility of post-quantum messaging but also reveal trade-offs between security models, deployment complexity, and performance. In particular, achieving post-quantum security while preserving E2E guarantees under an untrusted server model remains an open design challenge. Furthermore, comparatively little attention has been paid to the cost implications of different authentication and verification policies within such post-quantum E2E protocols, especially across heterogeneous hardware platforms. In this work, we address these challenges by presenting the design of a post-quantum end-to-end secure protocol for instant messaging applications under an untrusted relay model. The proposed construction leverages NIST-standardized lattice-based primitives, namely ML-KEM for key establishment and ML-DSA for authentication, and follows a Double-KEM design pattern combined with explicit context binding to derive a shared session key known exclusively to the communicating clients. The server participates only as an authenticated relay and never gains access to plaintext messages or session keys. The protocol design considers the formal structuring of a minimal post-quantum handshake, which is proved by explicit modeling under an unreliable relay by using symbolic verification of injective authentication and key secrecy. We also consider the formalization of transcript binding, the separation of roles and the evaluation of different verification policies. Finally, we experimentally characterize the computational cost of different verification policies and evaluate the performance of the proposed post-quantum end-to-end secure protocol for Instant Messaging Applications heterogeneous cloud-based architectures. Rather than claiming superiority over existing applications, our results provide practical insight into the feasibility and cost of deploying post-quantum end-to-end secure protocols within current instant messaging infrastructures.

https://doi.org/10.3390/cryptography10030028

The remainder of this paper is organized as follows. Section 2 reviews related work on secure and post-quantum instant messaging protocols. Section 3 introduces the cryptographic preliminaries used throughout the paper. Section 4 describes the protocol setting and the security model considered. Section 5 presents the design of the proposed post-quantum end-to-end secure protocol. Section 6 verify the security properties of our proposal. Section 7 reports the performance evaluation and results. Finally, Section 8 concludes the paper and outlines directions for future work.

2. Related Work

2.1. Secure Instant Messaging Protocols

The security of instant messaging (IM) protocols has been extensively studied in the context of end-to-end (E2E) encrypted communication systems. Modern messaging platforms such as Signal, WhatsApp, and Telegram employ bespoke cryptographic protocols whose security properties depend not only on the underlying primitives but also on their protocol-level design choices. Several works have demonstrated that subtle design decisions can lead to significant security implications, even when strong cryptographic algorithms are used. Schröder et al. [1] analyzed the usability and security properties of state-of-the-art secure messaging applications, highlighting the gap between theoretical security guarantees and practical user behavior. Jakobsen and Orlandi [14] showed that Telegram's MTProto protocol fails to achieve IND-CCA security due to its construction and composition choices, emphasizing that protocol design is as critical as primitive selection. Further analyses comparing Signal, WhatsApp, and Telegram can be found in [2,3], which illustrate differences in authentication mechanisms, trust assumptions, and threat resilience across platforms.

Formal verification techniques have also been applied to evaluate IM protocols. Miculan and Vitacolonna [15] analyzed MTProto 2.0 using the ProVerif tool and identified vulnerabilities related to unknown key-share (UKS) attacks under certain user behaviors, a class of attacks previously formalized in the context of authenticated key exchange protocols [13]. Similarly, Olivares Ménez et al. [19] employed AVISPA and Scyther to verify the security of Signal and MTProto 2.0 under bounded adversarial models, showing that security guarantees depend strongly on the assumed threat model and execution context. Broader surveys of cryptographic protocols used in IM applications confirm that protocol composition and trust assumptions remain a primary source of vulnerabilities [4,5].

2.2. Post-Quantum Messaging Applications

The emergence of post-quantum cryptography has motivated several efforts aimed at adapting secure messaging systems to withstand quantum adversaries. Early work by Duits [16] explored a post-quantum variant of the Signal protocol, demonstrating the feasibility of integrating lattice-based primitives into an E2E messaging framework. Subsequent studies have proposed or evaluated post-quantum secure messaging applications and experimental systems, including PQ Chat [18], and Vox Messenger [20], which incorporate post-quantum primitives to varying degrees. Other works have focused on revisiting secure messaging designs under post-quantum assumptions. Dursun et al. [17] discussed the challenges of constructing quantum-secure instant messaging applications and emphasized the impact of key sizes, performance overhead, and deployment constraints. While these efforts demonstrate that post-quantum messaging is achievable, they often differ in their security models, trust assumptions, and integration strategies, particularly regarding server involvement and E2E guarantees. Despite this progress, existing approaches either focus on adapting specific applications or on integrating post-quantum primitives into established protocols without explicitly addressing the design of a post-quantum end-to-end secure

protocol under an untrusted relay model. Moreover, comparatively little attention has been given to the cost implications of different authentication and verification policies within such protocols. In contrast, the present work focuses on the explicit design of a post-quantum E2E secure protocol that preserves strict confidentiality, mutual authentication, key agreement correctness, resistance to replay and unknown key-share attacks and server key secrecy against untrusted servers while characterizing the computational cost of alternative verification strategies.

2.3. Our Contribution

Modern instant messaging applications such as Signal, WhatsApp, Telegram, and the IETF Messaging Layer Security (MLS) framework address end-to-end security under production-oriented design constraints. These applications prioritize properties such as ratcheting-based forward secrecy, post-compromise security, multi-device synchronization, and large-scale deployment. The present work adopts a complementary perspective in the sense that the protocol design considers the formal structuring of a minimal post-quantum handshake, which is proved by explicit modeling under an unreliable relay by using symbolic verification of injective authentication and key secrecy. We also consider the formalization of transcript binding, the separation of roles and the evaluation of different verification policies. To clarify this distinction Table 1 summarizes key aspects, which is not intended to rank or evaluate these applications, but rather to highlight differences in key aspects. While production messaging applications integrate ratcheting, device management, and large-scale deployment mechanisms, the present work concentrates on the formalization and post-quantum characterization of a clean handshake primitive that can serve as a modular building block within broader messaging infrastructures.

Table 1. Key aspects of our proposal in comparison with related work.

Aspect	Signal/WhatsApp	Telegram	MLS	This Work
Cryptographic Basis	Classical/Hybrid PQ	Classical	Classical	Native PQ
Ratcheting	Yes	Limited	Yes	No
Formal Model Published	Partial	No public model	Limited	ProVerif model
Explicit Untrusted Relay Model	Implicit	No	Partial	Yes
Handshake Isolation	Integrated	Integrated	Integrated	Isolated

Table 1 also shows that from a practical deployment perspective, widely adopted messaging applications such as Signal and WhatsApp emphasize mature production features including ratcheting, post-compromise security, multi-device synchronization, and large-scale usability. Telegram follows a distinct architectural model with selective end-to-end encryption, while MLS targets scalable group key management. This kind of solutions embed key establishments within broader, feature-rich protocol ecosystems, resulting in increased architectural complexity, most of them utilizing classical cryptography. In contrast, the present work deliberately isolates a minimal post-quantum end-to-end handshake under an explicit untrusted relay model, trading advanced messaging functionality for modularity and formal analyzability. From a theoretical and quantum-resilience perspective, most deployed protocols rely on classical or hybrid constructions with incremental post-quantum integration. From the cryptographic bases side, our proposal instead adopts

a natively post-quantum design based exclusively on NIST-standardized lattice primitives (ML-KEM and ML-DSA), formally verified using ProVerif with explicit transcript binding, role separation, and Double-KEM composition. It is complemented by an empirical characterization of handshake cost under different verification policies. The resulting construction serves as a formally structured and quantum-resistant handshake building block that may be integrated into broader messaging architectures according to deployment needs.

Limitations

As we mentioned before, this work focuses on the design and verification of a post-quantum end-to-end handshake protocol under an untrusted relay model. In other words, the proposed protocol deliberately concentrates on establishing a post-quantum secure end-to-end construction. It does not incorporate ratcheting mechanisms, post-compromise security, multi-device synchronization, or group messaging functionality. These aspects are orthogonal extensions that can be layered on top of the core handshake construction and are left for future work. However, to maintain a clean and auditable core construction, we delimit the scope of the analysis in the sense that the formal verification is conducted in the Dolev–Yao adversary model using ProVerif. As customary in symbolic verification, the analysis assumes perfect cryptography and validates the protocol at the abstract model level. It does not constitute a computational security proof, nor does it include a reduction to standard hardness assumptions. Moreover, the symbolic model does not account for implementation-level aspects such as side-channel leakage, concrete parameter instantiations, cross-protocol interactions, or denial-of-service considerations. The verification focuses on structural properties of the handshake, namely injective mutual authentication and session-key secrecy. From the performance evaluation perspective, we characterize the computational cost of the handshake in controlled cloud-based environments. The experiments do not include measurements on mobile hardware, high-jitter network conditions, large-scale concurrent sessions, or energy and memory consumption. The objective of the evaluation is to provide a clear baseline for the cryptographic cost of the proposed construction.

3. Cryptographic Preliminaries

This section introduces the notation, cryptographic abstractions, and security assumptions used throughout the paper. We focus on high-level interfaces rather than internal algorithmic details, as the underlying primitives employed are standardized and well established.

3.1. Notation

Let $\mathbb{B}^n$ denote the set of all bit strings of length n. Concatenation of two bit strings x and y is denoted by $x \parallel y$. A probabilistic algorithm is denoted by $\xleftarrow{\$}$ to indicate uniform random sampling from its domain. Hash functions are modeled as deterministic functions $H : \{0,1\}^* \to \mathbb{B}^n$. We use HKDF to denote a standard hash-based key derivation function compliant with current cryptographic practice. Throughout the paper, we refer to a transcript as the ordered sequence of public messages exchanged during the execution of a protocol, including public keys, ciphertexts, and associated context information. Transcript binding refers to the derivation of cryptographic keys that are explicitly bound to this transcript. We denote by th the transcript hash computed as $th := TH(\cdot)$ over the ordered public handshake messages.

3.2. Post-Quantum Key Encapsulation Mechanism

We denote by ML-KEM the lattice-based key encapsulation mechanism standardized by NIST in FIPS 203 (formerly CRYSTALS-Kyber) [10]. ML-KEM consists of the following algorithms: $\mathsf{KeyGen}()$: outputs a public encapsulation key ek and a corresponding private decapsulation key dk. $\mathrm{Encaps}(ek)$: outputs a shared secret K and a ciphertext c. $\mathsf{Decaps}(dk, c)$: deterministically recovers the shared secret K from c. We assume that ML-KEM satisfies indistinguishability under chosen-ciphertext attacks (IND-CCA), providing confidentiality of the encapsulated secret even in the presence of adaptive adversaries.

3.3. Post-Quantum Digital Signatures

For authentication, we employ ML-DSA, the lattice-based digital signature scheme standardized by NIST in FIPS 204 (formerly CRYSTALS-Dilithium) [11]. ML-DSA provides the following algorithms: $\mathsf{KeyGen}()$: outputs a public verification key pk and a corresponding private signing key sk. $\mathrm{Sign}(sk, m)$: produces a digital signature σ on message m. $\mathrm{Verify}(pk, m, \sigma)$: verifies the validity of signature σ on message m. We assume that ML-DSA satisfies existential unforgeability under chosen-message attacks (EUF-CMA), ensuring the authenticity and integrity of signed protocol messages.

3.4. Authenticated Encryption and Key Derivation

Symmetric encryption of application data is performed using an authenticated encryption with associated data (AEAD) scheme. AEAD provides confidentiality and integrity guarantees for transmitted messages, while allowing additional public data to be authenticated but not encrypted. Session keys are derived using HKDF from shared secrets and protocol transcripts. The inclusion of transcript data in the key derivation process ensures that derived keys are bound to a specific protocol execution and context, preventing replay and unknown key-share attacks [13]. In the handshake, we instantiate transcript binding via a transcript-hash constructor $TH(\cdot)$, and use $H(th)$ as input to key confirmation tags.

3.5. Security Assumptions

The security of the proposed secure protocol relies on the following standard assumptions: ML-KEM is IND-CCA secure. ML-DSA is EUF-CMA secure. The employed hash functions and HKDF behave as pseudorandom functions. The AEAD scheme provides confidentiality and integrity under chosen-ciphertext attacks. No assumption is made regarding the trustworthiness of the server regarding message confidentiality; this aspect is addressed explicitly in the system and security model described in the next section.

4. Post-Quantum End-to-End Secure Protocol and Its Security Model

This section defines the post-quantum end-to-end secure protocol setting, trust assumptions, adversarial capabilities, and security objectives considered in this work. The model is tailored to instant messaging applications that operate over server-mediated infrastructures while requiring strict end-to-end (E2E) security guaranties in a post-quantum setting.

4.1. Post-Quantum End-to-End Secure Protocol Participants and Communication Model

The post-quantum end-to-end secure protocol consists of three types of entities: clients, a server, and a network adversary. *Clients:* Clients represent end users of the instant messaging application. They are assumed to be honest and to execute the prescribed protocol correctly. Each client possesses a long-term public–private key pair for authentication and may generate ephemeral keys during protocol execution. Clients aim to establish an end-to-end secure communication channel with other clients, such that session keys are known

exclusively to the communicating endpoints. *Server:* The server facilitates communication between clients by acting as a message relay. Its responsibilities include forwarding protocol messages, distributing public keys or certificates, and maintaining application-level connectivity. Crucially, the server is not trusted regarding message confidentiality or session key secrecy: it does not derive end-to-end session keys and does not decrypt application data. The server is modeled as an authenticated but untrusted relay and can arbitrarily relay, replay, reorder, or modify any protocol messages available to it. *Communication Channel:* All communication between clients is mediated by the server; no direct client-to-client channel is assumed. Client–server channels are assumed to provide authenticity and integrity, for example, through standard transport-layer protections, but they are not assumed to provide confidentiality. Consequently, all protocol messages forwarded by the server may be fully visible to an adversary controlling the server or the network.

4.2. Adversary and Threat Model

We consider a powerful network adversary with full control over the communication network, following a Dolev–Yao-adversary model. The adversary can intercept, modify, replay, delay, or drop messages exchanged between clients and the server. Furthermore, the server is assumed to be untrusted, so the adversary may compromise it, gaining access to all server-side state and observing or manipulating all messages processed by the server. Moreover, it is assumed to be computationally bounded but quantum-capable. In particular, the adversary is assumed to be unable to break the underlying cryptographic primitives employed by the protocol, namely ML-KEM under the IND-CCA security notion, ML-DSA under the EUF-CMA security notion, and the employed AEAD and hash-based key derivation mechanisms. No assumption is made regarding the confidentiality or integrity of server-side storage. When discussing automated symbolic verification, we adopt the standard Dolev–Yao adversary model, which assumes perfect cryptography and is consistent with standard protocol verification tools. However, client compromise is outside the scope of this work; clients are assumed not to be compromised during protocol execution. Denial-of-service attacks are also not addressed, as the focus of this work is on confidentiality, authentication, and key agreement properties.

4.3. Protocol Security Properties

The proposed secure protocol is designed to achieve the following security properties under the previously discussed adversary model. *End-to-End Confidentiality:* Application messages exchanged between two clients remain confidential to all third parties, including the server and any network adversary. In particular, neither the server nor an adversary controlling the server can derive end-to-end session keys or decrypt message contents. *Mutual Authentication:* Clients mutually authenticate each other during the channel establishment phase using post-quantum digital signatures, ensuring that session keys are bound to the identities of the communicating parties. *Key Agreement Correctness:* Upon successful completion of the protocol, both clients derive the same end-to-end session key, except with negligible probability. *Resistance to Replay and Unknown Key-Share Attacks:* Session keys are explicitly bound to the protocol transcript and participant identities, preventing replay attacks and unknown key-share (UKS) attacks. *Server Key Secrecy:* The server does not learn any information about end-to-end session keys, even if it deviates arbitrarily from the protocol or colludes with the network adversary. *Non-Goals:* The proposed design does not aim to provide post-compromise security, deniability, or group messaging functionality. Forward secrecy beyond the lifetime of a single session is also outside the scope of this work. These definitions establish the foundation upon which the end-to-end post-quantum secure protocol is constructed and analyzed in the subsequent sections.

5. Design of Our Post-Quantum End-to-End Secure Protocol

This section presents the design of an end-to-end post-quantum secure protocol tailored for instant messaging applications operating over a server-mediated infrastructure. The proposed construction adopts a client-server communication model in which the server is treated as an authenticated but untrusted relay regarding confidentiality: it forwards protocol messages and may enforce verification policies, yet it never obtains the end-to-end session key nor gains access to plaintext messages. The design combines long-term post-quantum digital signatures for identity authentication with ephemeral post-quantum key encapsulation mechanisms for session key establishment, and explicitly binds all cryptographic operations to a well-defined protocol context and session transcript. The resulting protocol preserves the end-to-end security guaranties expected from modern messaging systems while enabling a practical transition to NIST-standardized post-quantum cryptography within existing instant messaging architectures.

5.1. Post-Quantum End-to-End Secure Protocol Overview

The proposed secure protocol is designed for instant messaging applications that rely on a client–server architecture for message delivery, yet require end-to-end (E2E) confidentiality between communicating users. In contrast to traditional transport-layer secure channels, where the server typically participates as a cryptographic endpoint, the present design explicitly treats the server as an intermediary that is trusted only for availability and message routing, but not for confidentiality or key derivation. All cryptographic material required to derive the end-to-end session key is generated and processed exclusively by the communicating clients. It operates in two primary phases, conforming to a structure that is analogous to that of TLS. The *End-to-End Handshake and Key Establishment Phase* is responsible for mutual authentication of the communicating clients and for the derivation of a shared end-to-end session key using post-quantum cryptographic primitives. After the handshake is complete, the protocol moves on to the *Secure Messaging Phase*. During this phase, application messages are sent and received while being end-to-end encrypted using symmetric cryptography based on the session key. In both phases, the server functions as an untrusted relay and does not acquire access to plaintext data or cryptographic secrets. The design goals of our protocol are the following. It ensures post-quantum security by relying solely on NIST-standardized lattice-based primitives for public-key operations during session establishment, while restricting post-handshake communication to efficient symmetric cryptography. In addition, the protocol is designed to be compatible with existing instant messaging infrastructures, allowing servers to enforce optional verification policies for abuse mitigation without requiring access to session keys or plaintext data. The protocol deliberately avoids advanced features such as group messaging or continuous key ratcheting in order to focus on a clear and auditable E2E post-quantum channel design.

5.2. Cryptographic Roles and Key Material

The protocol involves three logical entities: two clients, denoted as Client 1 (initiator) and Client 2 (responder), and a server that relays messages between them. Each client possesses a long-term post-quantum digital signature key pair (pk_{Di}, sk_{Di}), instantiated using ML-DSA, which serves as the cryptographic identity of the client. These identity keys are assumed to be generated during an enrollment phase outside the scope of the protocol and are reused across multiple sessions. In contrast, each protocol execution employs fresh ephemeral key encapsulation key pairs (ek_i, dk_i) generated using ML-KEM, which are used exclusively for establishing session-specific shared secrets. Authentication within the protocol is achieved by binding ephemeral ML-KEM public keys to the protocol context using ML-DSA signatures. Specifically, each client signs a context-dependent statement

that includes its ephemeral encapsulation key and role information, producing a signature Sign_i that can be verified by the peer using the corresponding long-term public key pk_{Di}. The authenticity of pk_{Di} itself depends on an external trust mechanism. In this work, we assume a Trust-On-First-Use (TOFU) model as the default instantiation, whereby a peer's identity key is accepted upon first contact and subsequently pinned; alternative models such as certificate-authority-based authentication or trusted directories are compatible with the protocol but are not required for its core operation. The server does not possess any long-term or ephemeral private keys associated with the clients. Its cryptographic role is limited to optionally verifying identity credentials and signature bindings according to a configurable policy, and to relaying public keys, signatures, and ciphertexts between clients. Crucially, the server never receives ML-KEM decapsulation keys and never performs key derivation operations. As a result, all end-to-end session keys are derived exclusively by the clients, ensuring that the confidentiality of application messages does not depend on the trustworthiness of the server.

5.3. End-to-End Handshake and Key Establishment Phase

The end-to-end handshake establishes a shared session key exclusively between the two communicating clients, while all protocol messages are transported through a server that acts solely as an authenticated relay. The handshake combines authenticated advertisement of ephemeral post-quantum key encapsulation keys with a bidirectional key establishment procedure, ensuring that both parties contribute entropy to the final session key. All public-key operations are explicitly bound to a unique protocol context and session transcript to prevent misbinding, replay, and unknown key-share attacks. Throughout the handshake, the server may enforce configurable verification policies to control message forwarding and mitigate abuse; however, it never receives decapsulation keys, never performs key derivation, and never learns the resulting end-to-end session key. The complete message flow of the handshake is illustrated in Figure 1.

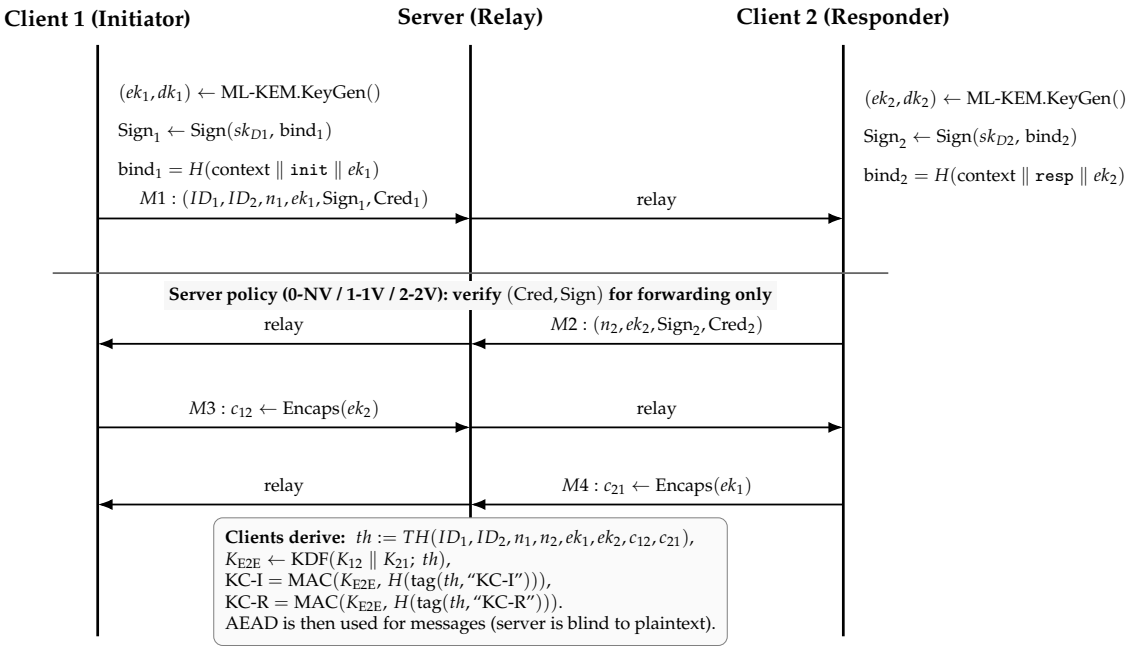


Figure 1. Minimal message flow of the end-to-end post-quantum handshake. The server relays protocol messages and may verify credentials/signatures for forwarding, but it never decapsulates and never derives the end-to-end session key.

5.3.1. Message $M1$: Initiator Key Advertisement and Authentication

Client 1 (the initiator) begins the handshake by generating an ephemeral ML-KEM key pair (ek_1, dk_1) and a fresh nonce n_1. It then constructs a protocol context encoding the application identifier, protocol version, the identities of both clients, and the nonce n_1. To authenticate its ephemeral encapsulation key and bind it to its role and session context, Client 1 computes the binding value $\text{bind}_1 = H(\text{context} \parallel \texttt{init} \parallel ek_1)$, and signs it using its long-term identity key sk_{D1}, producing the signature Sign_1. The message $M1 = (ID_1, ID_2, n_1, ek_1, \text{Sign}_1, \text{Cred}_1)$ is sent to the server, which relays it to Client 2. Depending on its configured policy, the server may verify Cred_1 and Sign_1 before forwarding, but it does not modify the message contents.

5.3.2. Message $M2$: Responder Key Advertisement and Authentication

Upon receiving M1, Client 2 retrieves the initiator's public verification key pk_{D1} using Cred_1 according to the selected trust model (CA-based, directory-based, or TOFU) and verifies Sign_1. If verification fails, the protocol is aborted. Client 2 then generates its ephemeral ML-KEM key pair (ek_2, dk_2) and a fresh nonce n_2. The protocol context is extended to incorporate n_2, ensuring freshness and explicit session binding. Client 2 computes $\text{bind}_2 = H(\text{context} \parallel \texttt{resp} \parallel ek_2)$, and signs it using its long-term identity key sk_{D2}, producing the signature Sign_2. The response message $M2 = (n_2, ek_2, \text{Sign}_2, \text{Cred}_2)$ is sent to the server and relayed to Client 1, again subject only to optional server-side verification for forwarding.

5.3.3. Message $M3$: First Key Encapsulation (Initiator to Responder)

After successfully verifying Sign_2, Client 1 performs an ML-KEM encapsulation using the responder's ephemeral key ek_2, obtaining a shared secret K_{12} and a ciphertext c_{12}. The ciphertext $M3 = c_{12}$ is transmitted to Client 2 via the server. The server observes only the ciphertext and does not possess the decapsulation key required to recover K_{12}.

5.3.4. Message $M4$: Second Key Encapsulation (Responder to Initiator)

Symmetrically, Client 2 encapsulates to the initiator's ephemeral key ek_1, producing a shared secret K_{21} and a ciphertext c_{21}. The ciphertext $M4 = c_{21}$ is relayed by the server to Client 1. Client 1 decapsulates using dk_1 to recover K_{21}. Together, the two encapsulation steps ensure that both parties contribute fresh entropy to the subsequent end-to-end session key derivation.

5.3.5. End-to-End Session Key Derivation

At the conclusion of the handshake, both clients hold the same pair of shared secrets (K_{12}, K_{21}). Let $th = TH(ID_1, ID_2, n_1, n_2, ek_1, ek_2, c_{12}, c_{21})$ denote the transcript hash over the ordered public handshake messages. The clients derive the final end-to-end session key as $K_{\text{E2E}} \leftarrow \text{KDF}(K_{12} \parallel K_{21};\ th)$. Only the two clients can compute K_{E2E}, as the server never performs decapsulation and never learns either shared secret. All subsequent application messages are protected using authenticated encryption under K_{E2E}. This bidirectional key establishment, based on two independent encapsulation secrets K_{12} and K_{21} combined through transcript-bound key derivation, constitutes a core design element of the protocol and is explicitly reflected in the symbolic verification model.

5.3.6. Key Confirmation

To confirm that both endpoints derived the same end-to-end key and to prevent role or direction confusions (e.g., reflection), the protocol performs symmetric key confirmation bound to the same transcript hash th with explicit domain separation. Concretely, the

initiator computes KC-I = MAC(K_{E2E}, $H(\text{tag}(th, \text{“KC-I”}))$), and the responder verifies it before returning KC-R = MAC(K_{E2E}, $H(\text{tag}(th, \text{“KC-R”}))$), which is then verified by the initiator. These role-separated confirmation tags ensure that both parties agree on the same transcript-bound session key before entering the secure messaging phase.

5.3.7. Symmetric Record Layer

From the end-to-end session key K_{E2E}, the two clients derive independent directional traffic keys using a key derivation function with explicit domain separation. In particular, K_{E2E} is expanded into $(K_{1\to2}, K_{2\to1})$, which are used exclusively for messages sent from Client 1 to Client 2 and vice versa. Application data are protected using an authenticated encryption with associated data (AEAD) scheme, such as AES-256-GCM or ChaCha20-Poly1305. Each direction maintains a monotonically increasing message counter from which nonces are deterministically derived, ensuring nonce uniqueness. The associated data binds each ciphertext to the protocol context and session transcript, providing integrity and preventing message replay, reordering, and cross-session substitution attacks. Throughout the record layer, the server remains oblivious to message contents and cannot forge, modify, or replay ciphertexts without detection.

5.4. Secure Messaging Phase

After the successful completion of the end-to-end handshake, both clients enter a steady-state secure messaging phase that relies exclusively on symmetric cryptography. In this phase, all application data are protected using authenticated encryption under keys derived from the end-to-end session key K_{E2E}, and no further public-key or lattice-based operations are required. This design ensures that the per-message computational cost remains comparable to that of existing end-to-end encrypted messaging applications, while preserving post-quantum security for the initial key establishment. From K_{E2E}, the clients derive two independent directional traffic keys, $K_{1\to2}$ and $K_{2\to1}$, using a key derivation function with explicit domain separation. Directional keys prevent reflection attacks and ensure that compromise in one communication direction does not immediately affect the other. Each client maintains a monotonically increasing message counter for every direction, which is used to deterministically derive nonces, thereby guaranteeing nonce uniqueness without requiring additional randomness during the messaging phase. Messages are encrypted using an authenticated encryption with associated data (AEAD) scheme, such as AES-256-GCM or ChaCha20-Poly1305. For a plaintext message m, the sender computes $C \leftarrow \text{AEAD.Enc}(K_{i\to j}, \mathsf{nonce}, m;\ \text{AD})$, where nonce is derived from the direction-specific message counter and AD denotes associated data. The associated data binds each ciphertext to the session context and transcript information (e.g., protocol identifier, client identifiers, and message counter), ensuring integrity and preventing replay, reordering, and cross-session substitution attacks. Upon receiving a ciphertext, the recipient recomputes the corresponding nonce and verifies the AEAD tag using the appropriate directional key. If verification fails, the message is discarded. Because the server never learns K_{E2E} or any derived traffic key, it cannot decrypt message contents nor generate valid ciphertexts. Consequently, during the secure messaging phase the server remains a transport intermediary only, while confidentiality and integrity are enforced end-to-end between the communicating clients.

5.5. Deployment Considerations

The proposed Post-Quantum End-to-End Secure Protocol focuses on a clean and formally analyzable post-quantum end-to-end handshake under an untrusted relay model. Its integration into a full-fledged messaging system requires consideration of several system-level aspects that are orthogonal to the cryptographic core described as follows. *Identity and*

Trust Establishment: The protocol assumes the existence of persistent long-term public keys for communicating parties. The mechanism by which such keys are initially authenticated (e.g., trust-on-first-use, certificate- based infrastructures, or transparency mechanisms) is intentionally left unspecified. The handshake construction itself is agnostic to the particular trust bootstrap mechanism, provided that the authenticity of long-term public keys can be established. *Integration with Server-Mediated Architectures:* Our design is compatible with typical server-mediated messaging infrastructures in which the server acts as an authenticated relay without access to plaintext messages or session keys. The relay is explicitly modeled as untrusted regarding end-to-end confidentiality, and therefore no modification of the server trust assumptions is required beyond message forwarding and optional authentication checks. *Migration from Classical to Post-Quantum Settings:* In practice, existing messaging applications may adopt post-quantum mechanisms incrementally. The proposed handshake could be deployed as a standalone post-quantum channel establishment mechanism or, if desired, combined with classical primitives in a hybrid setting. Such deployment choices do not alter the structural correctness of the handshake construction. *Extensibility:* Advanced messaging features such as ratcheting mechanisms, multi-device synchronization, or group messaging protocols can be layered on top of the established session key without modifying the handshake's internal logic. The present work deliberately isolates the post-quantum handshake primitive to enable modular integration into broader messaging frameworks.

6. Formal Security Verification of Our Post-Quantum End-to-End Secure Protocol Using ProVerif

We performed a symbolic verification of the Post-Quantum End-to-End Secure Protocol using ProVerif 2.05. The verification is conducted under the standard Dolev–Yao adversary model (perfect cryptography), as commonly adopted by protocol verification tools. It means, cryptographic primitives are modeled as ideal constructors and equations, and the adversary is given full control of the network. This kind of model captures the essential structure and security-relevant elements of the protocol while abstracting from implementation-level details in accordance with common practice in automated protocol verification. To support the security claims, we use a symbolic model of the proposed end-to-end handshake.

6.1. Post-Quantum End-to-End Secure Protocol Modeling

As we mentioned before, authentication of ephemeral ML-KEM encapsulation keys is achieved through ML-DSA signatures over context-bound binding values. Each signature explicitly binds the sender's role (initiator or responder), ephemeral encapsulation key, and protocol context, thereby preventing replay, misbinding, and cross-protocol attacks. The inclusion of fresh nonces in the authenticated context ensures session uniqueness and freshness across protocol executions. Moreover, the bidirectional (double-KEM) key establishment guarantees that both parties contribute fresh entropy to the final end-to-end session key. An adversary who compromises only one ephemeral decapsulation key cannot recover the full session key, as both shared secrets (K_{12}, K_{21}) are required for key derivation. The final session key is derived as a function of these secrets and a transcript hash $th = TH(ID_1, ID_2, n_1, n_2, ek_1, ek_2, c_{12}, c_{21})$, which binds the key to the complete ordered sequence of public handshake messages. In addition, the protocol employs explicit symmetric key confirmation using role-separated tags (KC-I and KC-R) computed over the same transcript hash th. This confirmation step ensures that both endpoints have derived the same transcript-bound session key and prevents reflection and

direction-confusion attacks. As a result, successful completion of the handshake implies mutual agreement on the peer identity, session parameters, and end-to-end session key before entering the secure messaging phase. In this sense, the modeled protocol comprises two honest clients (initiator and responder) communicating over a public channel and an *untrusted relay* that forwards messages but may arbitrarily relay, replay, reorder, delay, inject, or modify traffic. This is captured by a replicated relay process that simply receives and outputs attacker-visible messages on the public channel. As we said before, we consider a Dolev–Yao adversary model. It controls the network, can compose and decompose messages according to available public constructors, and can exploit protocol interleavings and replays. Therefore, cryptographic primitives are assumed unbreakable except through their formal interfaces.

6.2. Modeled Cryptographic Abstractions

In a general way, post-quantum cryptographic primitives are modeled symbolically as ideal constructors. In particular, the model includes (i) Lattice-based key encapsulation via abstract encapsulation and decapsulation functions, (ii) Digital signatures used to authenticate ephemeral key material, (iii) A key derivation function, and (iv) Message authentication codes for symmetric key confirmation. Perfect cryptography is assumed, as is customary in symbolic verification. In addition, ProVerif explicitly captures the bidirectional (Double-KEM) structure of the handshake by modeling the two independent encapsulation secrets K_{12} and K_{21}, their combination via transcript-bound key derivation, and the resulting end-to-end session key. Specifically speaking, the symbolic model faithfully represents the authenticated Double-KEM handshake. Two independent encapsulation steps generate shared secrets K_{12} and K_{21}, which are combined through a transcript-bound key derivation function. The transcript hash explicitly binds participant identities, nonces, ephemeral public keys, and ciphertexts. Symmetric key confirmation is modeled using role-separated MACs (KC-I/KC-R) to prevent reflection and role-confusion attacks. It means the symbolic model captures the handshake structure described in Section 5.3. It includes (i) authenticated advertisement of ephemeral ML-KEM public keys via signatures on context-bound binding terms; (ii) transcript binding via a transcript-hash constructor $TH(\cdot)$ over the ordered public handshake messages; and (iii) symmetric key confirmation bound to the same transcript hash and separated by role using explicit tags (KC-I/KC-R). Concretely, the end-to-end key is derived as $th = TH(ID_1, ID_2, n_1, n_2, ek_1, ek_2, c_{12}, c_{21})$, $K_{\mathrm{E2E}} = \mathrm{KDF}(K_{12}, K_{21}, th)$, and key confirmation is modeled as two MACs over $H(\mathrm{tag}(th, \cdot))$ with distinct role tags. The KEM functionality is represented by constructors for ciphertext and shared key with explicit randomness, together with the single decapsulation equation $\mathrm{decaps}(dk(ek), \mathrm{encap_c}(ek, r)) = \mathrm{encap_k}(ek, r)$. Long-term signing keys are private, and signatures are modeled in constructor-only style, with verification enforced by equality checks against the expected signed term.

6.3. Verified Properties

Automated symbolic verification is employed to validate the proposed handshake's security properties under the Dolev–Yao adversary model. The protocol's security properties are defined in Section 4.3, but they are not independent in the symbolic setting. Instead, they can be categorized into two main groups that can be formally verified with ProVerif. Firstly, we verify *Mutual authentication and agreement (injective)* by encoding role-specific authentication events for both parties and prove injective correspondences: whenever the initiator completes the handshake, there exists a unique matching responder authentication event, and symmetrically for responder completion. These injective guarantees establish

mutual authentication, inherently prevent replay-based impersonation attacks, and ensure agreement on peer identities and session parameters, including the derived end-to-end session key. Additionally, capturing key agreement correctness and resistance to unknown key-share (UKS) attacks through explicit binding of identities and protocol transcripts. Secondly, we verify *Session-key secrecy* by providing that there is no reachable trace in which a session key is marked as established while the adversary learns it, modeled by the non-reachability of attacker(k) jointly with an event session_key(k). Given that the server is modeled as untrusted and does not possess cryptographic secrets, this secrecy guarantee directly implies end-to-end confidentiality of the established session keys regarding both the network adversary and a potentially compromised server. In the context of the automated symbolic verification, these two groups of properties collectively encompass all security properties that are specified in Section 4.3.

6.4. Formally Verified Security Properties

Security properties are expressed in ProVerif as correspondence and secrecy queries over explicitly defined protocol events. The protocol is modeled in the applied pi-calculus, where authentication is represented through role-specific events marking critical execution points (e.g., authentication and successful completion), and secrecy is expressed through non-reachability of attacker knowledge. On the one hand, authentication is encoded through role-specific events (`c1_auth`, `c2_auth`) and completion events (`c1_end`, `c2_end`). Injective correspondence queries are used to ensure that each successful completion of one party corresponds to a unique matching authentication event of the peer. The following injective correspondence queries are verified, which formally guarantee: Mutual authentication between the communicating parties, agreement on peer identities and session parameters, agreement on the derived end-to-end session key, resistance to replay attacks (via injectivity), and resistance to unknown key-share (UKS) attacks through explicit transcript and identity binding.

```
query x,y,n1,n2,ek1,ek2,k;
event(c1_end(x,y,n1,n2,ek1,ek2,k))
==> inj-event(c2_auth(x,y,n1,n2,ek1,ek2)).

query x,y,n1,n2,ek1,ek2,k;
event(c2_end(x,y,n1,n2,ek1,ek2,k))
==> inj-event(c1_auth(x,y,n1,n2,ek1,ek2)).
```

On the other hand, end-to-end confidentiality is formalized through a secrecy query over the established session key. In this sense, the protocol marks successful key establishment using the event `session_key(k)` and secrecy is expressed as non-reachability of adversarial knowledge of that key as follows. This query guarantees that no execution trace exists in which a session key is marked as successfully established while being learned by the adversary. Since the server is modeled as an untrusted relay and does not possess decapsulation keys, this secrecy property directly implies end-to-end confidentiality of session keys even under server compromise. Table 2 summarizes the correspondence between protocol security properties defined in Section 4.3 and their formalization as ProVerif queries.

```
query k:bitstring;
attacker(k) && event(session_key(k)).
```

Table 2. Protocol security properties formally verified by ProVerif queries.

Protocol Security Property	Formalization in ProVerif	Result
Mutual Authentication	Injective correspondence between completion and peer authentication events	Verified
Key Agreement Correctness	Agreement on derived session key and transcript parameters	Verified
Replay Resistance	Implied by injective authentication	Verified
Unknown Key-Share Resistance	Transcript- and identity-bound agreement	Verified
End-to-End Confidentiality	Non-reachability of attacker(k) with session_key(k)	Verified
Server Key Secrecy	Session-key secrecy under untrusted relay model	Verified

6.5. Verification Results

ProVerif verifies all stated queries for the modeled handshake: both injective authentication correspondences hold, and the session-key secrecy query is satisfied. In particular, the model validates that the derived end-to-end key is bound to the full transcript hash *th*, and that the symmetric key confirmation messages (KC-I/KC-R), separated by role tags, are consistent with the intended transcript-bound agreement semantics. Throughout the protocol, the server never receives ML-KEM decapsulation keys and never performs decapsulation or key derivation. Consequently, it cannot derive the end-to-end session key, even if it observes all protocol messages. Server-side verification modes (0-NV, 1-1V, 2-2V) affect only forwarding policy and abuse mitigation; they do not influence cryptographic secrecy. After the handshake, all application messages are protected using AEAD under directional traffic keys derived from the end-to-end session key. Nonce construction based on monotonically increasing counters guarantees uniqueness, while associated data binds each ciphertext to the session context and message order. Any modification, injection, or replay of ciphertexts by the server or an external adversary is detected during decryption.

As we have been mentioning, our post-quantum end-to-end secure protocol considers the formal structuring of a minimal post-quantum handshake, which is proved by explicit modeling under an unreliable relay by using symbolic verification of injective authentication and key secrecy. We also consider the formalization of transcript binding and separation of roles and the evaluation of different verification policies. Table 3 summarizes verified properties that our proposal achieves in comparison to related work.

Table 3. Formal verification of instant messaging applications.

Protocol	Automated Verification Tools Used	Formally Verified Properties
Signal/PQ-Signal variants	Symbolic analysis (Tamarin/ProVerif-style) [21,22]	Authentication, secrecy, (post-quantum) forward secrecy variants, stateful aspects (handshake + ratchet)
Telegram MTProto 2.0	ProVerif [15]	Authentication, secrecy, integrity, PFS; rekeying analysis and reported UKS issue in some settings
MLS (TreeKEM)	Symbolic and machine-checked analyses [23]	Group key agreement, epoch secrecy/ authentication, stateful group semantics
This Work	ProVerif 2.05	Injective mutual authentication/parameter agreement; session-key secrecy under Dolev–Yao

7. Performance of Our Post-Quantum End-to-End Secure Protocol

This section evaluates the practical performance of the proposed end-to-end post-quantum secure protocol under realistic deployment conditions. The objective of the experimental study is not to benchmark individual cryptographic primitives in isolation, but rather to assess the end-to-end cost of establishing and operating a post-quantum secure messaging protocol in a client-server-client architecture, where the server acts exclusively as an authenticated relay. The evaluation focuses on execution time and CPU overhead across heterogeneous hardware platforms and verification modes, reflecting the constraints of both low-resource devices and server-grade environments.

7.1. Experimental Setup

The experimental setup simulates an instant messaging scenario in which two clients establish an end-to-end post-quantum secure protocol through a relay server. The implementation follows the protocol design described in Section 5, including authenticated key advertisement, bidirectional ML-KEM-based key establishment, transcript-bound key derivation, and a symmetric secure messaging phase based on AEAD encryption. All experiments were conducted using ML-KEM-768 and ML-DSA-3, corresponding to NIST security level 3.

To capture architectural diversity, experiments were conducted across multiple hardware configurations spanning both ARM and x86 platforms, including low-resource environments representative of mobile or edge devices, as well as server-grade instances. All experiments were executed on Linux-based systems in cloud environments, ensuring reproducibility and controlled conditions. The experimental campaign comprised a total of 12,648 independent test runs, covering a good quantity of combinations of hardware platform and server verification mode. Each test run executed a full protocol cycle, including the end-to-end handshake and a representative secure messaging exchange. This repetition was designed to reduce noise and capture stable performance characteristics rather than isolated best- or worst-case behavior.

7.2. Metrics and Methodology

The evaluation focuses on protocol-level performance metrics that are directly relevant to real-world deployment of instant messaging applications. The following metrics were collected during each test run. *Total Execution Time*: the wall-clock time required to complete a full protocol cycle, including authenticated key exchange, bidirectional encapsulation, key derivation, and secure message transmission. *CPU Time*: the effective CPU processing time consumed by the protocol execution, measured independently of I/O or scheduling delays. Measurements were collected using lightweight system monitoring instrumentation, allowing fine-grained tracking of CPU utilization without interfering with protocol execution. Experiments were conducted under three server-side verification modes: 0-NV (no verification), 1-1V (single-sided verification), and 2-2V (mutual verification). These modes affect only the server's forwarding policy and do not alter the end-to-end cryptographic guarantees of the channel. The analysis is intentionally descriptive. Although distributional plots are used to visualize the variability and concentration of measurements, no strong assumptions about strict normality are imposed. Instead, the focus is on consistency, stability, and comparative trends across platforms and verification levels.

7.3. Results

The experimental results demonstrate that the proposed post-quantum end-to-end secure protocol achieves low and stable performance overhead across all evaluated configurations. In particular, both total execution time and CPU consumption remain well

within practical bounds for instant messaging applications, even under the most demanding verification mode. Tables 4 and 5 report representative mean values and variability for total execution time and CPU time, respectively, aggregated by platform class and verification mode. Across all tested architectures, increasing the server verification level introduces a modest and predictable overhead, with no abrupt performance degradation. In particular, even under mutual verification (2-2V), CPU overhead remains below approximately 15 ms on server-grade hardware and around 10 ms on low-resource ARM platforms, indicating that the additional verification cost is manageable in practice. For completeness and reproducibility, the full set of experimental results across all evaluated platforms and configurations is reported in Appendix A. Figure 2 illustrates the distribution of CPU usage across all test runs. The observed values exhibit a strong concentration at low utilization levels, with limited dispersion across platforms and verification modes. This behavior suggests that the computational cost our proposal is both predictable and stable, a desirable property for interactive messaging applications that must maintain responsiveness under varying load conditions. Overall, the results confirm that the proposed design enables a practical transition to post-quantum end-to-end security without imposing prohibitive computational overhead. Once the handshake is completed, the protocol operates exclusively in a symmetric cryptographic mode using AEAD encryption, ensuring that ongoing message exchange incurs costs comparable to those of existing end-to-end encrypted messaging applications. These findings support the feasibility of deploying the proposed secure protocol in real-world instant messaging applications across heterogeneous device classes.

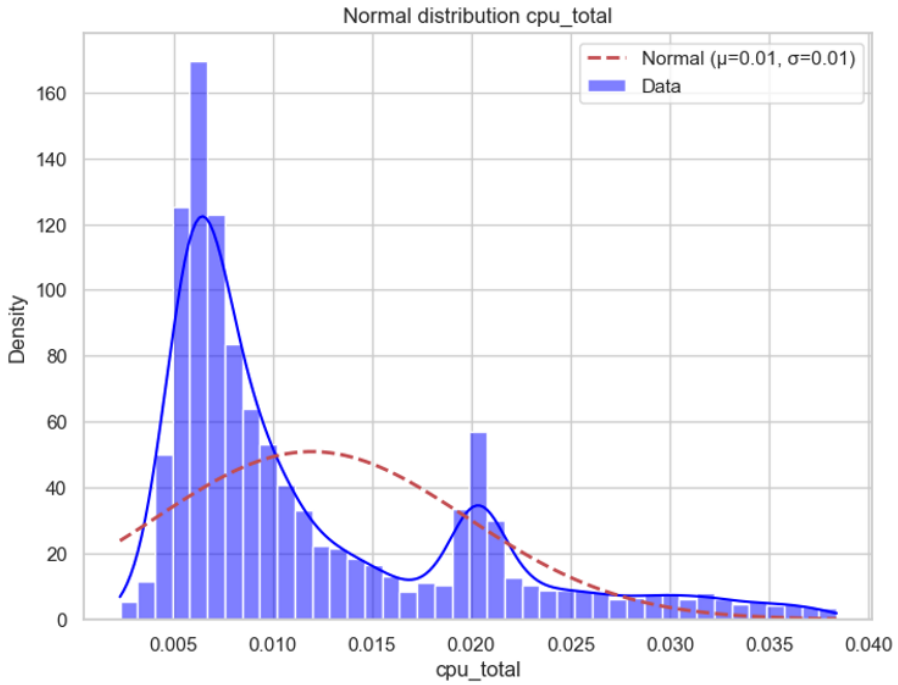


Figure 2. The distribution of CPU usage across all test runs shows normal fit (μ = 0.01, σ = 0.01), the data histogram and the blue line as the kernel density estimation.

Table 4. Total execution time across representative platforms (mean ± standard deviation, in seconds).

Platform	0-NV	1-1V	2-2V
ARM (low-resource)	0.0488 ± 0.0049	0.0485 ± 0.0045	0.0504 ± 0.0054
ARM (server-grade)	0.0491 ± 0.0051	0.0488 ± 0.0050	0.0492 ± 0.0049
x86 (server-grade)	0.0526 ± 0.0075	0.0523 ± 0.0069	0.0530 ± 0.0079

Table 5. CPU time overhead across representative platforms (mean ± standard deviation, in seconds).

Platform	0-NV	1-1V	2-2V
ARM (low-resource)	0.0086 ± 0.0061	0.0087 ± 0.0051	0.0106 ± 0.0067
ARM (server-grade)	0.0093 ± 0.0050	0.0090 ± 0.0050	0.0103 ± 0.0049
x86 (server-grade)	0.0119 ± 0.0092	0.0133 ± 0.0085	0.0136 ± 0.0096

8. Conclusions

This work presents the design of a clear and auditable post-quantum end-to-end secure protocol tailored to instant messaging applications under an untrusted relay model and operating over a server-mediated infrastructure. The design deliberately emphasizes context binding, mutual authentication, and deployability within existing instant messaging infrastructures. It prioritizes a clean and auditable post-quantum handshake over advanced messaging features such as ratcheting, post-compromise security, multi-device synchronization, or group messaging, which remain natural directions for future extensions. In contrast to transport-layer secure channels, where the server is typically a cryptographic endpoint, our construction treats the server as an authenticated relay that is not trusted for confidentiality. By combining NIST-standardized lattice-based primitives—ML-KEM for key establishment and ML-DSA for authentication—with context binding and a Double-KEM key agreement pattern, the protocol establishes an end-to-end session key known exclusively to the communicating clients and remains robust against active network adversaries and server observation.

From the formal security verification perspective, our results hold under the Dolev-Yao adversary model, which can be considered as customary for symbolic verification. It focuses on the abstract handshake structure rather than on implementation-level aspects. In addition, the symbolic verification establishes injective mutual authentication and session-key secrecy under these modeling assumptions. In this sense, we can conclude that the model focuses on the handshake and its key establishment and confirmation logic. It does not attempt to model side-protocol leakage, denial-of-service, implementation bugs, or the full record-layer message flow beyond the established session key. Beyond the security verification of our protocol, we experimentally characterized the practical cost of deploying such a post-quantum end-to-end secure protocol under heterogeneous hardware conditions and configurable verification policies. The results let us conclude that total execution time and CPU overhead remain low and stable across ARM and x86 platforms, and that stronger server-side verification modes introduce only modest and predictable additional overhead. These findings support the feasibility of adopting post-quantum end-to-end secure protocols in real-world instant messaging applications without imposing prohibitive performance costs during session establishment. It is achieved because after handshake, the protocol operates solely in a symmetric record layer based on AEAD encryption, ensuring that steady-state messaging performance remains comparable to conventional end-to-end encrypted deployments.

It is important to mention that our post-quantum end-to-end secure protocol intentionally prioritizes clarity and deployability over advanced secure messaging features such as asynchronous ratcheting, postcompromise security, deniability, or group messaging support. These aspects are orthogonal to the core goal addressed here—namely, establishing a clean and auditable post-quantum end-to-end protocol under an untrusted relay model—and are left for future investigation. Considering the aforementioned, we can say that the results provide a concrete and reproducible foundation for the deployment of post-quantum end-to-end secure channels in server-mediated messaging applications. Finally, we can say that promising directions include formal symbolic verification of the full protocol, integration with post-quantum ratcheting mechanisms for long-lived sessions,

and expanded evaluation on constrained mobile and IoT devices as well as under realistic network conditions and server load.

Author Contributions: Conceptualization, A.F.D.A.-L., K.A.D.-V. and G.G.-G.; Methodology, G.G.-G. and E.S.-R.; software, H.A.O.A.; validation, G.G.-G. and E.S.-R.; formal analysis, E.S.-R.; investigation, A.F.D.A.-L. and K.A.D.-V.; data curation, E.S.-R.; writing—original draft preparation, A.F.D.A.-L., K.A.D.-V. and H.A.O.A.; writing—review and editing, G.G.-G. and K.A.D.-V.; funding acquisition, G.G.-G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Instituto Politécnico Nacional under grant number MULTI-2026-0012-M4.

Data Availability Statement: For reproducibility, all verification queries were successfully proved. The complete ProVerif model (`pqhandshake.pv`) and the full verification logs are available by contacting the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Complete Experimental Results

This appendix reports the complete set of experimental measurements collected during the evaluation described in Section 7. The table includes all tested hardware configurations, verification modes, and descriptive statistics. These results are provided for transparency and reproducibility and are not required for the main performance discussion.

Table A1. Total Time and CPU Time.

Descriptive Statistics	Total Time			CPU Time		
	0-N V	1-1 V	2-2 V	0-N V	1-1 V	2-2 V
Mean	0.009477	0.006424	0.005393	0.009434	0.007740	0.006383
Median	0.008452	0.005053	0.004661	0.008808	0.006130	0.006222
Standard Deviation	0.004002	0.003567	0.002773	0.003787	0.003930	0.002754
Count	249	328	270	249	328	270
aarch64-2cpu-352MB-Linux						
Mean	0.083395	0.144950	0.050006	0.009553	0.013913	0.009996
Median	0.047948	0.114601	0.048012	0.005446	0.012197	0.007195
Standard Deviation	0.061567	0.322922	0.004750	0.007864	0.007598	0.005777
Count	218	246	224	218	246	224
x86_64-2cpu-1901MB-Linux						
Mean	0.059812	0.059785	0.059421	0.022090	0.025132	0.023459
Median	0.055732	0.054911	0.055627	0.016286	0.018870	0.018448
Standard Deviation	0.013059	0.014669	0.012929	0.015752	0.017794	0.015896
Count	308	293	320	308	293	320
x86_64-2cpu-3827MB-Linux						
Mean	0.058157	0.058332	0.059512	0.020564	0.023251	0.023086
Median	0.053190	0.051646	0.053920	0.014383	0.015271	0.016109
Standard Deviation	0.011927	0.014542	0.014491	0.014795	0.017527	0.017145
Count	313	329	298	313	329	298
x86_64-2cpu-7811MB-Linux						
Mean	0.054675	0.054242	0.056227	0.016505	0.017899	0.019151
Median	0.050718	0.049923	0.051392	0.011026	0.012198	0.012889
Standard Deviation	0.008906	0.009228	0.010875	0.010962	0.011513	0.013222
Count	319	315	333	319	315	333

Table A1. *Cont.*

Descriptive Statistics	Total Time			CPU Time		
	0-N V	1-1 V	2-2 V	0-N V	1-1 V	2-2 V
x86_64-4cpu-15779MB-Linux						
Mean	0.053643	0.055376	0.055223	0.014205	0.018927	0.017856
Median	0.049268	0.050865	0.051359	0.008603	0.012171	0.011930
Standard Deviation	0.009408	0.010917	0.009509	0.011726	0.013707	0.011873
Count	451	490	467	451	490	467
aarch64-2cpu-1845MB-Linux						
Mean	0.048779	0.048541	0.050448	0.008629	0.008720	0.010610
Median	0.047467	0.046922	0.048097	0.005312	0.006488	0.007382
Standard Deviation	0.004882	0.004514	0.005396	0.006109	0.005113	0.006694
Count	223	200	239	223	200	239
aarch64-2cpu-3834MB-Linux						
Mean	0.049308	0.048924	0.050485	0.008947	0.009114	0.010717
Median	0.046662	0.047153	0.048188	0.005657	0.006388	0.007357
Standard Deviation	0.005305	0.004837	0.005550	0.006193	0.005654	0.006493
Count	213	221	238	213	221	238
aarch64-2cpu-7812MB-Linux						
Mean	0.048886	0.048847	0.050402	0.008771	0.009157	0.010194
Median	0.046573	0.046470	0.048233	0.005359	0.006593	0.007038
Standard Deviation	0.005242	0.004859	0.005100	0.006378	0.005577	0.006075
Count	229	211	208	229	211	208
aarch64-4cpu-15770MB-Linux						
Mean	0.049162	0.049131	0.049611	0.009282	0.010324	0.010008
Median	0.048291	0.048309	0.048331	0.006221	0.007185	0.007023
Standard Deviation	0.004928	0.004895	0.004975	0.006449	0.006121	0.006126
Count	507	517	518	507	517	518
x86_64-8cpu-31714MB-Linux						
Mean	0.052594	0.052276	0.053005	0.011901	0.013335	0.013620
Median	0.048117	0.048064	0.048164	0.007393	0.008894	0.008856
Standard Deviation	0.007450	0.006880	0.007926	0.009229	0.008512	0.009606
Count	548	551	546	548	551	546
aarch64-8cpu-31684MB-Linux						
Mean	0.049120	0.048793	0.049203	0.009167	0.010286	0.010031
Median	0.048371	0.048286	0.048359	0.006164	0.007108	0.007099
Standard Deviation	0.005089	0.004995	0.004850	0.006389	0.006122	0.006003
Count	569	571	568	569	571	568

References

1. Schröder, S.; Donko-Huber, M.; Wind, D.; Rottermanner, C. When Signal Hits the Fan: On the Usability and Security of State-of-the-Art Secure Mobile Messaging. In *European Workshop on Usable Security*; IEEE: New York, NY, USA, 2016.
2. Dahiru, I.; Atiku, A.; Naibi, R. Secure Messaging: Analysis of Signal Protocol Implementation in WhatsApp and Signal. *J. Digit. Innov. Contemp. Res. Sci. Eng. Technol.* **2018**, *6*, 63–72.
3. Bogos, C.-E.; Mocanu, R.; Simion, E. *A Security Analysis Comparison Between Signal, WhatsApp and Telegram*; Cryptology ePrint Archive, IACR: Santa Barbara, CA, USA, 2023; p. 071. Available online: https://eprint.iacr.org/2023/071 (accessed on 21 October 2025).
4. Miron, A.-D.; Cristian, A.; Simion, E.; Andries, S. *A Survey on the Security Protocols Employed by Mobile Messaging Applications*; Cryptology ePrint Archive, IACR: Santa Barbara, CA, USA, 2022; Paper 2022/088. Available online: https://eprint.iacr.org/2022/088 (accessed on 21 October 2025).

5. Johansen, C.; Mujaj, A.; Arshad, H.; Noll, J. The Snowden Phone: A Comparative Survey of Secure Instant Messaging Mobile Applications. *Secur. Commun. Netw.* **2021**, *2021*, 9965573. [CrossRef]
6. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 1996.
7. Paar, C.; Pelzl, J. *Understanding Cryptography: A Textbook for Students and Practitioners*; Springer: Berlin/Heidelberg, Germany, 2010.
8. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
9. NIST. Post-Quantum Cryptography: Selected Algorithms. National Institute of Standards and Technology. 2022. Available online: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 (accessed on 21 October 2025).
10. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation. 2023. Available online: https://pq-crystals.org/kyber/index.shtml (accessed on 20 October 2025).
11. Bai, S.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation (Version 3.1). 2023. Available online: https://pq-crystals.org/dilithium/index.shtml (accessed on 20 October 2025).
12. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3*; Request for Comments (RFC) 8446; Internet Engineering Task Force: Fremont, CA, USA, 2018. Available online: https://www.rfc-editor.org/rfc/rfc8446 (accessed on 21 October 2025). [CrossRef]
13. Blake-Wilson, S.; Menezes, A. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *Public Key Cryptography (PKC)* ; Springer: Berlin/Heidelberg, Germany, 1999; pp. 154–170.
14. Jakobsen, J.; Orlandi, C. On the CCA (In)Security of MTProto. In Proceedings of the CCS'16: 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 113–116.
15. Miculan, M.; Vitacolonna, N. Automated Symbolic Verification of Telegram's MTProto 2.0. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021), Virtual, 6–8 July 2021; pp. 185–197. [CrossRef]
16. Duits, I. The Post-Quantum Signal Protocol: Secure Chat in a Quantum World. Ph.D. Dissertation, University of Twente, Enschede, The Netherlands, 2019.
17. Dursun, A.-F.; Seyhan, K.; Aydin, B.-K.; Akleylek, S. Quantum Secure Instant Messaging: Revisited. In Proceedings of the II International Conference on Information Security: Problems and Prospects, Baku, Azerbaijan, 25 November 2022.
18. Post-Quantum. PQ Chat. 2023. Available online: https://post-quantum.com/messaging/index.html (accessed on 21 October 2025).
19. Ménez, G.O.O.; Delgado-Vargas, K.A.; Bernstein-Mery, F.; Gallegos-García, G. Security Verification of Instant Messaging Cryptographic Protocols. In *Telematics and Computing*; Springer: Cham, Switzerland, 2023; pp. 418–435.
20. Vox Messenger. Encryption Core. 2018. Available online: https://vox-messenger.app/encryption/ (accessed on 21 October 2025).
21. Cremers, C.; Horvat, M.; Hoyland, J.; Scott, S. A Comprehensive Symbolic Analysis of the Signal Protocol. In Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Paris, France, 26–28 April 2017.
22. Bhargavan, K.; Jacomme, C.; Kiefer, F.; Schmidt, R. Formal Verification of the PQXDH Post-Quantum Key Agreement for End-to-End Messaging. In Proceedings of the USENIX Security Symposium, Philadelphia, PA, USA, 14–16 August 2024.
23. Wallez, R.; Protzenko, J. TreeKEM: A Modular Machine-Checked Symbolic Security Analysis of MLS. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 12–15 May 2025.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.