Ilyazh-Web3E2E: A Post-Quantum Hybrid Protocol for Forward-Secure Decentralized Messaging

Ilyas Zhaisenbayev
Independent Researcher
Email: ilyaszhaisenbayev@gmail.com
Version 0.3 (2025-08-19)

***Abstract*—This paper specifies Ilyazh-Web3E2E, a cryptographic protocol designed to provide robust, multi-layered security for peer-to-peer communication in the Web3 era. It addresses the dual threat of classical and quantum adversaries by implementing a hybrid authenticated key exchange (AKE) combining classical (X25519) and post-quantum (Kyber-768) primitives. The protocol ensures forward and post-compromise security through a Double Ratchet algorithm and guarantees confidentiality and integrity via AES-256-GCM, achieving IND-CCA security. We define its formal threat model, security goals, and cryptographic specification, positioning it as a practical and formally motivated candidate for next-generation secure messaging.**

***Index Terms*—Post-Quantum Cryptography, End-to-End Encryption, Double Ratchet, Hybrid Encryption, Kyber, Secure Messaging.**

I. Introduction

The proliferation of decentralized technologies (Web3) necessitates cryptographic protocols that are not only secure against current threats but also resilient against future quantum adversaries. Existing secure messaging protocols, while robust, face the impending challenge of quantum computers capable of breaking classical public-key cryptography [2]. This paper introduces Ilyazh-Web3E2E, a protocol designed to address this challenge. Its architecture is founded on the principle of **trust through transparency**, eschewing proprietary "black-box" designs in favor of a verifiable composition of standardized, publicly scrutinized cryptographic primitives. By combining the classical Diffie-Hellman function X25519 with the NIST-standardized post-quantum KEM, CRYSTALS-Kyber [4], it provides a robust hybrid key exchange. This is coupled with a Double Ratchet mechanism inspired by the Signal protocol [3] to provide strong forward and post-compromise security. The contributions of this paper are threefold:

- We present a complete specification for a hybrid, post-quantum authenticated key exchange.
- We detail a Double Ratchet integration providing forward and post-compromise security.
- We provide a comparative analysis against established protocols and outline a path to a secure implementation.

II. Related Work

The design of Ilyazh-Web3E2E builds upon decades of research in secure messaging and cryptography. Key influences include the Signal Protocol [3], the NIST PQC standardization process [4], and academic work on hybrid encryption [9].

III. Threat Model and Security Goals

A. Threat Model

The protocol is designed to be secure against a powerful, **active network adversary**. The adversary can read, modify, inject, replay, and delete packets at will. Our formal model focuses on these network-based attacks. We explicitly note that local attacks, such as **side-channel analysis** or a **compromised Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)**, are outside the scope of this protocol's formal model. However, we mandate in Section VI-D that any practical implementation must include countermeasures against these threats.

B. Security Goals

- **Confidentiality (IND-CCA):** The content of messages is computationally indistinguishable from random to any party other than the intended recipient.
- **Integrity & Authenticity:** It is computationally infeasible for an adversary to modify or forge messages without detection.
- **Forward Secrecy (FS):** The compromise of long-term keys does not compromise past messages.
- **Post-Compromise Security (PCS):** The protocol can "heal" from a session state compromise.
- **Post-Quantum Security (PQS):** Confidentiality is maintained against a quantum adversary.

IV. Cryptographic Specification

The protocol is divided into phases, each building upon the last to establish and maintain a secure channel.

A. Cryptographic Primitives and Rationale

The default suite combines classical and post-quantum algorithms.

TABLE I
DEFAULT CRYPTOGRAPHIC SUITE

Component	Specification	Rationale
KEM (Classical)	X25519	High-performance, widely adopted classical security
KEM (PQ)	Kyber-768	NIST Level 3 PQC standard for post-quantum resistance
AEAD	AES-256-GCM	Standard for high-efficiency IND-CCA encryption
KDF	HKDF-SHA384	Robust derivation of cryptographically separate keys
Signature	Ed25519	High-performance signatures for strong authentication

B. Phase 1: Authenticated Key Exchange (AKE)

The initial handshake establishes a mutually authenticated shared secret. The resulting shared secret (ss) is derived as:

$$ss = \text{HKDF-Extract}(salt, \text{X25519}(sk_A, pk_B) \,\|\, \text{Kyber.Decaps}(sk_{A,p} \quad (1)$$

where $\|$ denotes concatenation.

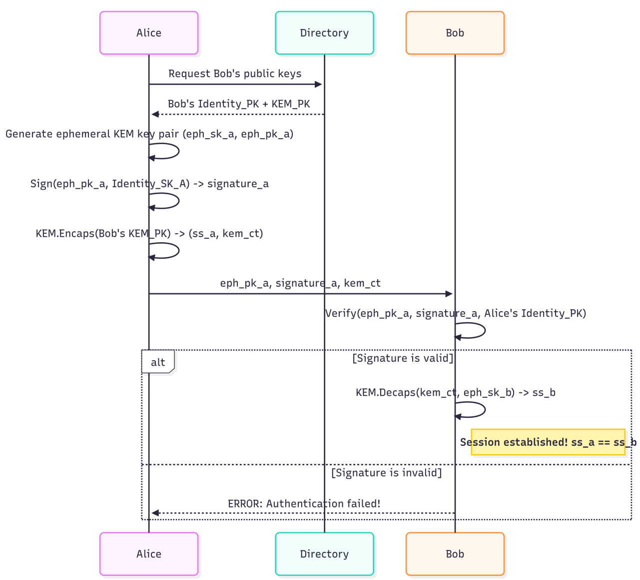

Fig. 1. Session Establishment Flow (AKE)

C. Phase 2: Double Ratchet Messaging

The protocol uses a standard Double Ratchet algorithm [6] to manage session keys, providing both FS and PCS for every message.

D. Wire Format and Associated Data (AAD)

A fixed binary format is specified. All unencrypted header fields are authenticated as Associated Data.

V. SECURITY MODEL AND PROOF SKETCHES

A. Confidentiality and Integrity (IND-CCA)

We define security via the standard IND-CCA game. *Proof Sketch:* We prove IND-CCA security by reduction. Assume a

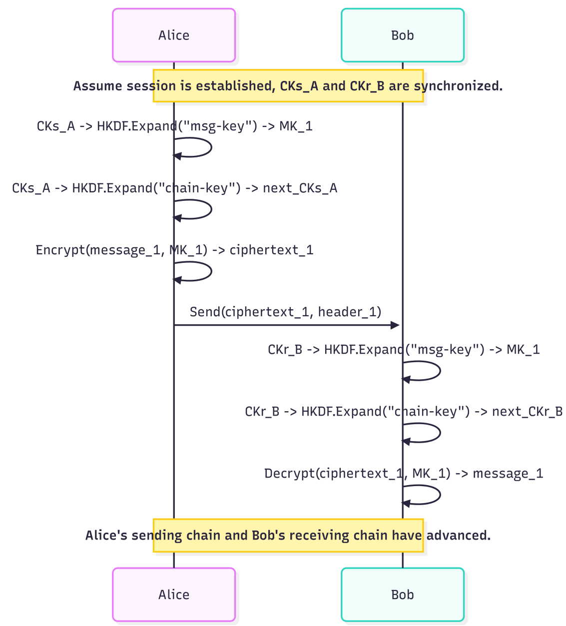

Fig. 2. Symmetric-key ratchet step for deriving a Message Key

PPT adversary $\mathcal{A}$ wins the game. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break either the IND-CCA security of AES-256-GCM or the IND-CPA security of the hybrid KEM. This contradicts the security assumptions of the underlying primitives. *Note:* While these sketches provide a strong argument, a full, machine-checked proof using a formal verification tool such as Tamarin or ProVerif is left as future work.

VI. IMPLEMENTATION AND PERFORMANCE ANALYSIS

A. Implementation

A reference implementation is provided in Python to demonstrate the protocol's logic. For production, a rewrite in a memory-safe language such as Rust is required.

B. Preliminary Benchmarks

The following benchmarks are preliminary results from the non-optimized Python PoC on a consumer laptop (Intel Core i7).

TABLE II
PRELIMINARY PERFORMANCE BENCHMARKS (PYTHON POC)

Metric	Approximate Value
Handshake Latency (Full AKE)	150-200 ms
AEAD Throughput (1MB message)	20-25 MB/s

C. Comparison with Existing Protocols

D. Security Considerations

E. Nonce Management

To prevent key reuse in AES-GCM, the 96-bit nonce MUST be unique for each message. The specified structure is the con-

TABLE III
PROTOCOL COMPARISON

Feature	Ilyazh-Web3E2E	Signal	TLS 1.3	MLS
PQ Status	Hybrid	No	No	No
Forward Secrecy	Yes	Yes	Yes	Yes
PCS	Yes	Yes	No	Yes
Handshake Latency	150-200 ms	50-100 ms	50-100 ms	High
Ciphertext Overhead	16 B (tag)	16 B (tag)	16 B (tag)	Moderate

catenation of a **64-bit random prefix** and a **32-bit monotonic counter**.

F. Limits and Invariants

An implementation MUST enforce session limits. A session MUST be re-established after 2^{32} messages or 24 hours.

VII. CONCLUSION

The Ilyazh-Web3E2E protocol provides a complete, high-security specification for post-quantum E2E encrypted messaging. By composing standardized primitives, it achieves strong formal security goals, positioning it as a practical candidate for next-generation secure applications. Future work includes machine-checked formal verification in Tamarin or ProVerif.

ACKNOWLEDGEMENTS

The author thanks Professor Henry Corrigan-Gibbs of MIT for valuable guidance. The design reflects lessons from the MIT 6.1600 course [1].

APPENDIX

A. Wire Format Specification

All protocol messages use the following binary format. Fixed-length fields are big-endian encoded. **Associated Data**

TABLE IV
WIRE FORMAT STRUCTURE

Field	Size (bytes)	Description
Version	1	Protocol version (0x03)
Suite ID	2	Crypto suite identifier
Sequence Num	8	Monotonic message counter
Nonce	12	AEAD nonce (64-bit prefix + 32-bit counter)
Header Len	2	Length of encrypted header
Encrypted Header	*var*	CBOR-encoded ratchet headers
Ciphertext	*var*	AEAD ciphertext + 16-byte tag

(AAD) = Version || Suite ID || Sequence Num

B. Protocol Invariants & Limits

- **Nonce Structure:** Nonce = `0xRRRRRRRRRRRRRRRR || 0x000000CC`
  - `R`: 64-bit cryptographically secure random (per ratchet step)
  - `CC`: 32-bit monotonic counter (reset to 0 on ratchet)
- **Rekeying:** Mandatory after:
  - 2^{20} messages (per chain)
  - 24 hours of continuous use
- **Session Limits:** Terminate session after:
  - 2^{32} total messages
  - 7 days of activity

C. Handshake (AKE Phase)

- **Alice's Identity Key:** `1f2c3d4e...` (Ed25519 private key, 32 bytes)
- **Bob's Kyber Ciphertext:** `8956a7b8...` (Kyber-768 ciphertext, 1088 bytes)
- **Shared Secret Output:** `234bcd5e...` (64 bytes)

D. Message Encryption

- **AAD:** `030001000000000000001` (Version 0x03, Suite 0x0001, Seq 1)
- **Nonce:** `4e3291d850a43b00000001` (64-bit random + 32-bit counter)
- **Plaintext:** `48656c6c6f2057656233` ("Hello Web3")
- **Ciphertext:** `89ab12cd...` (plaintext length + 16 bytes tag)

Full reproducible test vectors available in reference implementation.

E. Side-Channel Attacks

- **Threats:** Timing attacks on KEM operations, memory access patterns
- **Countermeasures:**
  - Constant-time implementations for Kyber/X25519
  - Hardware-isolated memory for ratchet states
  - Zeroization of sensitive buffers

F. Random Number Generation

- **Threats:** Low-entropy seeding, VM snapshot attacks
- **Countermeasures:**
  - Hybrid entropy sources (HW RNG + OS entropy)
  - Periodic reseeding (every 100 operations)
  - Forward-secure RNG design

G. Supply Chain Risks

- **Threats:** Compromised dependencies, malicious hardware
- **Countermeasures:**
  - Reproducible builds with auditable dependencies
  - Hardware roots of trust for key generation
  - Multiple KEM diversity (e.g., add Dilithium)

H. Operational Security

- **Critical Requirement:** All countermeasures MUST be production-level
- **Verification:** Use formal methods (Cryptol, SAW) for primitives

This section provides key Python code snippets from the reference implementation, illustrating the core cryptographic logic of the Ilyazh-Web3E2E protocol. These examples are for educational purposes and serve to clarify the specification detailed in the main body of the paper.

I. Phase 1: Authenticated Key Exchange (AKE)

The handshake involves a combination of classical (X25519) and post-quantum (Kyber-768) key exchange primitives to derive a shared secret.

```python
def initiate_handshake(self,
    bob_identity_public_bytes):
    # ... (code to generate ephemeral keys and KEM
        keypair) ...

    # Encapsulate a shared secret using Bob's KEM
        public key
    kemalg = self.kem_algorithm
    kem_public_key_bob = oqs.KeyEncapsulation(kemalg
        , public_key=bob_identity_public_bytes)
    kem_ciphertext, shared_secret = self.
        kem_private_key.encap_secret(
        kem_public_key_bob.export_public_key())

    # Perform classical DH key exchange with a
        temporary key
    dh_output = self.ephemeral_private_key.exchange(
        x25519.X25519PublicKey.from_public_bytes(
            kem_public_key_bob.export_public_key())
    )

    # Derive the initial root key and chain key from
        the DH output and KEM shared secret
    initial_shared_secret = dh_output +
        shared_secret
    root_key, chain_key = self._kdf_rk(
        initial_shared_secret, b'')

    # ... (further steps to construct the handshake
        message and store session state)
    return message
```

Listing 1. IlyazhProtocol.initiate_handshake

J. Phase 2: Double Ratchet Messaging

The Double Ratchet algorithm is responsible for evolving session keys for each message, providing forward and post-compromise security.

```python
def encrypt_message(self, session_id, message,
    associated_data=b''):
    # ... (session validation) ...

    # Perform DH Ratchet if needed (new chain or
        session start)
    if session["message_numbers"]["send"] == 0:
        new_ratchet_private = x25519.
            X25519PrivateKey.generate()
        session["sending_ratchet_private"] =
            new_ratchet_private

        # ... (key exchange and root key derivation)
            ...

    # Derive a message key and update the chain key
        for the next message
    message_key, new_chain_key = self._kdf_chain(
        session["sending_chain"], b'')
    session["sending_chain"] = new_chain_key
    session["message_numbers"]["send"] += 1

    # ... (nonce, header, and AEAD encryption using
        message_key) ...
    return ciphertext_payload
```

Listing 2. IlyazhProtocol.encrypt_message

K. Key Derivation Functions (KDF)

These functions use HKDF-SHA384 to securely derive new, cryptographically separate keys from shared secrets and other inputs, preventing key reuse.

```python
def _kdf_chain(self, chain_key, associated_data):
    # HKDF-Expand-SHA384
    h = hmac.HMAC(chain_key, hashes.SHA384(),
        backend=default_backend())
    h.update(associated_data)
    prk = h.finalize()

    hkdf = HKDF(
        algorithm=hashes.SHA384(),
        length=64,
        salt=b'',
        info=b'chain_key_message_key',
        backend=default_backend()
    )
    # Return two 32-byte keys: message_key and
        new_chain_key
    return tuple(hkdf.derive(prk)[i:i+32] for i in
        range(0, 64, 32))
```

Listing 3. IlyazhProtocol._kdf_chain

References

[1] H. Corrigan-Gibbs and N. Zeldovich, "6.1600: Foundations of Computer Security," Fall 2023. MIT. [Online]. Available: https://61600.csail.mit.edu/2023/

[2] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484-1509, 1997.

[3] T. Marlinspike and M. Oxley, "The Signal Protocol," IETF, Internet-Draft draft-signal-protocol-01, 2017.

[4] National Institute of Standards and Technology (NIST), "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," NISTIR 8413, 2022.

[5] T. Marlinspike and M. Oxley, "The X3DH Key Agreement Protocol," Signal, Nov. 2016. [Online]. Available: https://signal.org/docs/specifications/x3dh/

[6] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 441-456.

[7] R. Barnes et al., "The Messaging Layer Security (MLS) Protocol," RFC 9420, IETF, July 2023.

[8] R. Avanzi et al., "CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.02)," NIST PQC Submission, 2021.

[9] D. Stebila, N. P. Smart, and S. C. C. Quintino, "Post-quantum key exchange for the internet and the open quantum safe project," in *Dependable and Secure Computing*, 2018, pp. 1-8.

[10] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)," RFC 8949, IETF, Dec. 2020.