Technical Documentation: Post-Quantum OpenID Connect

Team Name: Trojan Valkyries

Date: 9 February 2026

Project: Post-Quantum Secure OpenID Connect using KEMTLS

---

1. System Architecture Overview

1.1 High-Level Design

This project implements a fully functional **Post-Quantum OpenID Connect (OIDC)** system. It replaces the classical TLS transport layer with a **KEMTLS** (Key Encapsulation Mechanism Transport Layer Security) protocol and secures the application-layer Identity Tokens with **NIST-standardized Post-Quantum Signatures**.

The system is built on a **Node.js** runtime and consists of three distinct entities interacting over TCP sockets:

1. **User Agent (UA):** The client application acting as the browser. It initiates the OIDC flow and manages the KEM-based secure sessions.
2. **Relying Party (RP):** The service provider. It verifies the Post-Quantum Identity Tokens before granting access to protected resources.
3. **Identity Provider (IDP):** The authentication server. It authenticates the user and issues `ML-DSA-65` (Dilithium3) signed Identity Tokens.

1.2 Technology Stack

- **Runtime:** Node.js (v20+)
- **Transport Security:** Custom KEMTLS implementation using `ML-KEM-768` (Kyber).
- **Cryptography Engine:**
    - **KEM:** C-compiled WebAssembly (`wrapper.wasm`) for high-performance key encapsulation.
    - **Signatures:** `dilithium-crystals-js` library for ML-DSA operations.

- **Protocol:** Custom TCP framing for KEMTLS handshakes; JSON-based messaging for OIDC flows.

---

2. Cryptographic Design Choices & Rationale

2.1 Transport Layer: KEMTLS (ML-KEM-768)

Instead of using standard TLS 1.3 (which relies on RSA/ECC signatures for authentication), we implemented **KEMTLS**.

- **Algorithm: ML-KEM-768 (Kyber)**.
- **Design Rationale:** In a post-quantum environment, digital signatures (like Dilithium) are significantly larger in size (approx. 3-4KB) compared to classical signatures. Using them in every handshake (as in standard PQ-TLS) introduces high bandwidth overhead.
- **Mechanism:** Our KEMTLS implementation authenticates the server implicitly. The server provides a long-term KEM public key. If the client can encapsulate a shared secret to this key and the server can decapsulate it, the server is authenticated. This removes the need for a heavy handshake signature, reducing latency and bandwidth.

2.2 Application Layer: Digital Signatures (ML-DSA-65)

The OpenID Connect Identity Token (ID Token) is the core credential in this system. It must be verifiable and non-repudiable.

- **Algorithm: ML-DSA-65 (Dilithium3)**.
- **Design Rationale:** We selected Dilithium3 because it is one of the primary algorithms selected by NIST for standardization. It offers a "Strong Security" level (equivalent to AES-192) which balances security and performance better than the larger Dilithium5 or the slower SPHINCS+.
- **Implementation:** The ID Token follows the JWT (JSON Web Token) structure but uses a custom `alg` header (`ML-DSA-65`). The signature is encoded in Base64 and appended to the payload.

3. Benchmarking Methodology & Performance Results

3.1 Methodology

We evaluated the performance of the system by measuring the wall-clock time of cryptographic operations during a live end-to-end authentication flow.

- **Environment:** Local test environment (Intel Core i5/i7, Node.js Runtime).
- **Metrics:** Latency (ms) and Message Size (bytes).

3.2 Latency Results

The following data was captured during a successful login flow (User alice):

Operation	Algorithm	Recorded Latency (ms)	Description
KEMTLS Handshake	ML-KEM-768	**73.75 ms** (Avg)	Average of UA to RP (83ms) and UA to IDP (64ms).
Token Signing	ML-DSA-65	**46.33 ms**	Time taken by IDP to generate keys and sign the JWT.
Token Verification	ML-DSA-65	**33.76 ms**	Time taken by RP to verify the Dilithium signature.

Analysis: The KEMTLS handshake latency is acceptable for a secure connection establishment. The verification time (33ms) is highly efficient, ensuring that the User Experience (UX) at the Relying Party remains smooth.

3.3 Message Size & Protocol Overhead

Post-Quantum cryptography introduces larger key and signature sizes. Our measurements for a single ID Token are as follows:

Component	Size (Bytes)	Notes
JWT Header	31 bytes	Standard JSON header.
JWT Payload	101 bytes	Contains `sub`, `iss`, `aud` claims.
Dilithium Signature	**3,544 bytes**	Base64 encoded `ML-DSA-65` signature.
Total Token Size	**~4.9 KB**	Significantly larger than RSA tokens (~1KB).

3.4 Comparison with Reference Benchmarks

We compared our results with the reference paper *"Post-quantum electronic identity"* (Schardong et al., 2022).

- **Signature Size:** The reference paper lists the raw Dilithium3 signature size as **3,293 bytes**. Our implementation resulted in **3,544 bytes**. This slight increase is due to the Base64 encoding required for safe JSON transport, which is an expected overhead in web protocols.
- **Handshake Efficiency:** Standard PQ-TLS (using Dilithium for handshake auth) would require transmitting roughly **5-6KB** of data during the handshake. Our **KEMTLS** implementation avoids transmitting the server signature entirely, keeping the handshake lightweight.

4. Conclusion

This project successfully demonstrates a functional Post-Quantum OpenID Connect system. By utilizing **KEMTLS** for transport, we mitigated the bandwidth issues associated with post-quantum signatures. By using **Dilithium3** for the ID Token, we ensured the system is compliant with upcoming NIST standards. The benchmarks confirm that while token sizes are larger, the computational latency (33-46ms) remains well within practical limits for real-world deployment.