

PassmanAI.org: A Hybrid Post-Quantum OpenID Connect Identity Provider for Secure and Backward-Compatible Identity Federation

Ashish Ku. Jena, Department of Computer Science and Engineering, GIET University, Gunupur, Odisha, India, ashishkumarjena1437@gmail.com

Swayam Sopnic Nayak, Department of Computer Science and Engineering, GIET University, Gunupur, Odisha, India, swayamsopnic@gmail.com

Sushobhan Pal, Department of Computer Science and Engineering, GIET University, Gunupur, Odisha, India, sushobhanpalofficial@gmail.com

Abstract: Federated identity protocols such as OAuth 2.0 and OpenID Connect (OIDC) rely on RSA and ECDSA, both of which are broken in polynomial time by Shor’s algorithm on a sufficiently capable quantum computer. Full migration of the OIDC token format itself to NIST-standardized post-quantum algorithms is blocked by the immaturity of the surrounding ecosystem. This paper presents PassmanAI, a deployed-prototype OIDC identity provider that resolves this tension architecturally rather than waiting on standardization. PassmanAI confines post-quantum cryptography (ML-KEM-768, ML-DSA-65, and the standards-track X-Wing hybrid KEM) to the boundary between the user’s browser and the identity provider, while the externally-facing ID and access tokens remain classically signed (RS256), preserving drop-in compatibility with the unmodified OAuth/OIDC library ecosystem. We describe the system’s architecture, key management, and threat model, and evaluate the cost of this design through cryptographic primitive benchmarks, an end-to-end login latency and through-put comparison against a classical control (Ed25519), and a direct event-loop lag diagnostic. Under concurrent load, the post-quantum login path shows p95 latency growing roughly linearly with concurrency, consistent with a queue already near saturation, while the classical control retains throughput headroom across the same range.

Keywords: Post-Quantum Cryptography, OpenID Connect, OAuth 2.0, ML-KEM, ML-DSA, X-Wing, Identity Federation, Hybrid Cryptography

1. Introduction

Federated identity protocols are the load-bearing infrastructure of consumer and enterprise authentication on the web. OAuth 2.0 and OpenID Connect (OIDC) in particular underpin” sign in with” flows across the vast majority of consumer-facing applications, and depend on RSA and ECDSA to sign access tokens, ID tokens, and client assertions. Both primitives are broken in polynomial time by Shor’s algorithm on a cryptographically relevant quantum computer (CRQC). The completion of NIST’s Post-Quantum Cryptography (PQC) standardization process in 2024 finalizing ML-KEM (FIPS 203) [1] and ML-DSA (FIPS 204) [2] provides the algorithmic foundation for migration, but application-layer identity protocols remain largely unmigrated in practice. The obstacle is not the absence of algorithms; it is ecosystem readiness. The JOSE (JSON Object Signing and Encryption) specification that underlies JWT signing has no finalized algorithm identifiers for post-quantum signatures, and the relying-party libraries third-party developers already depend on NextAuth.js, Authlib, Passport.js and do not recognize ML-DSA signatures. A naive full-stack migration that signs the externally-facing ID token with ML-DSA would silently break every existing OIDC client library in production today. This paper presents PassmanAI, a deployed-prototype OIDC identity provider that resolves this tension architecturally.

Rather than waiting for JOSE standardization or asking every relying party to upgrade their libraries, PassmanAI splits the authentication journey into two cryptographically distinct phases at a single, explicit trust boundary: a post-quantum-protected "front door" between the user's browser and the identity provider, and a classically-signed "back door" between the identity provider and the relying party's backend. This design trades complete end-to-end post-quantum coverage of the bearer token for immediate, unmodified compatibility with the existing OAuth/OIDC ecosystem a trade-off we argue is justified by the short lifetime of the affected tokens, and one we evaluate empirically in this paper. Here, in this paper, we present the architecture of a hybrid PQC/classical OIDC identity provider, including its key management model, composite hybrid key encapsulation for application-layer data sharing, and a refresh-token replay-detection mechanism. We develop NPM and PyPI integration packages that enable third-party applications to adopt the system without implementing PKCE, discovery, or token validation, following established library patterns. An explicit threat model maps adversarial capabilities to corresponding security mechanisms and identifies residual risks. The system is evaluated against a matched classical control using cryptographic overhead, end-to-end latency, concurrency tests, and event-loop diagnostics to identify performance bottlenecks. The paper covers related work, threat model, architecture, implementation, evaluation, results, limitations, and conclusions.

2. Literature Survey

Schardong et al. [3] present the first full-stack empirical evaluation of post-quantum OAuth 2.0 and OpenID Connect, replacing classical TLS key exchange and JWT signatures with Kyber and NIST finalist signature algorithms across the entire protocol stack, implemented as an extension of the open-source pyoidc and pyjwkest libraries. Their results show that post-quantum OIDC achieves comparable or better performance than classical OIDC in low-latency network conditions, but that the TLS handshake accounts for roughly fifty percent of total request duration in high-latency scenarios establishing that network and handshake effects, rather than raw cryptographic computation, dominate the cost of full-stack PQC migration. A subsequent dissertation [4] extends this line of work toward broader identity-management and fiduciary frameworks. Müller and Oupický [5] conduct an empirical evaluation of post-quantum SAML Single Sign-On, measuring bandwidth overhead for several hybrid signature and key-encapsulation combinations. They report that a Dilithium (ML-DSA) and Kyber (ML-KEM) hybrid increases bandwidth by approximately $3.56\times$ relative to a classical baseline, that SPHINCS+ with Kyber increases it by $7.93\times$, and that Classic McEliece configurations increase it by approximately $76\times$, rendering the latter unsuitable for production SAML deployments. Their work also identifies that the HTTP Redirect binding becomes unworkable under post-quantum signature sizes, motivating a shift toward the Artifact binding.

PassmanAI differs from both lines of work in a specific, structural way: rather than migrating the externally-visible protocol artifact (the JWT or the SAML assertion) to post-quantum algorithms, it confines post-quantum cryptography to the internal authentication boundary and leaves the externally-facing token format unchanged. This is a narrower claim than full-stack migration, but it directly targets the ecosystem-compatibility barrier incomplete JOSE standardization and unmodified relying-party libraries that both [3] and [5] identify as an open obstacle to deployment, rather than the network/bandwidth overhead that both papers focus on measuring. On the formal

side, Fett, Ku^usters, and Schmitz [6] provide the most comprehensive formal security analysis of OAuth 2.0 to date, establishing precise definitions of authorization, authentication, and session integrity within an expressive web model. The threat model presented in Section III of this paper is deliberately scoped at a practitioner/architectural level rather than a formal-proof level; we cite [6] as the appropriate benchmark for the rigor a future formal treatment of this architecture would need to meet. Table 1 shows Threat Model and its map with each adversary capability considered to the specific mechanism in the architecture that addresses it. Consistent with the formal-versus-architectural distinction drawn in Section II, this analysis is a structured mapping of capability to mechanism, not a machine-checked proof in the style of Fett et.al. [6].

Table 1. Threat Model Summary

ID	Threat	Mitigation
T1	Authorization code interception	PKCE (S256)[7] binds the authorization code to the legitimate client.
T2	Harvest-now-decrypt-later attack on login	Hybrid ML-KEM-768/X25519 (X-Wing) with ML-DSA-65 authentication.
T3	Harvest-now-decrypt-later attack on ID/access tokens	TLS 1.3 using the X25519MLKEM768 hybrid key exchange.
T4	Compromised relying-party database	Only SHA-256 hashes of bearer tokens are stored.
T5	Refresh-token replay	Single-use refresh-token rotation with family-wide revocation.
T6	Master-password compromise	Argon2id protects locally stored ML-DSA, ML-KEM and X25519 private keys.
T7	Consent forgery or downgrade	Consent is bound to the authenticated session established after PQC login.
T8	Future RS256 signature forgery	Short-lived tokens and aggressive signing-key rotation reduce exposure.

T2, T3, and T8 together illustrate the central design trade-off of this paper. T2 and T3 are now both addressed: front-door authentication is protected by application-layer post-quantum cryptography, and all traffic, including the classically-signed back-door token, is protected against future decryption by transport-layer hybrid key agreement (Section IV-G). T8 isolates the one property that remains genuinely classical and unaddressed by either mechanism — the long-term unforgeability of the RS256 sig-nature itself and is the residual risk this paper treats as a stated limitation (Section VIII) rather than a solved problem.

3. System Architecture

The system architecture is designed around three key goals: protect user secrets against future quantum attacks, maintain compatibility with existing OAuth/OIDC systems without requiring modifications, and avoid storing private keys on the server to prevent the creation of a high-value security target.

3.1. The Hybrid Trust Boundary

Fig. 1 illustrates the system's central architectural decision. The authentication journey is split into two cryptographically distinct phases at an explicit boundary inside the PassmanAI platform. In the PQC front door, the user's browser and PassmanAI's servers establish a post-quantum-protected channel using the standards-track X-Wing hybrid KEM [8] (ML-KEM-768 combined with X25519 via the IETF-specified domain-separated SHA3-256 combiner, implemented via the official XWing export of the @noble/post-quantum library [9]), and the user proves possession of their private key by signing a server-issued nonce with ML-DSA-65 a zero-knowledge authentication step that never transmits the password itself. In the classical back door, once PQC authentication completes, PassmanAI issues a standard OAuth 2.0 authorization code and, on token exchange, an RS256-signed ID token using a 3072-bit RSA key, ensuring the externally-facing artifact is verifiable by any unmodified OIDC client library.

3.2. Composite Hybrid Encapsulation for Application Data

Beyond the login challenge, PassmanAI uses the same X-Wing construction for application-layer payload encapsulation (e.g., vault sharing), combining ML-KEM-768 and X25519 shared secrets through the standardized combiner to derive an AES-256-GCM key. Because X-Wing's combiner is defined to inherit security from whichever of its two component schemes remains unbroken, this provides defense-in-depth: a future cryptanalytic break of ML-KEM-768 alone, or of Curve25519 alone, does not by itself compromise the derived key. As of this writing, draft-connolly-cfrg-xwing-kem remains an IETF Internet-Draft rather than a published RFC; the construction is therefore standards-track rather than finalized.

3.3 Developer Integration Layer

Third-party relying parties integrate PassmanAI through published client SDKs rather than implementing OIDC discovery and PKCE themselves the same integration pattern used by established identity SDKs such as google-auth-library. The @pass-manai/nextauth package (NPM) is a thin configuration wrapper over nextauth that auto-configures OIDC discovery and enforces PKCE and state checks; the passmanai package (PyPI) is built directly on Authlib's OAuth2Session and delegates PKCE generation and validation to it. Both SDKs are open source, which we consider a pre-condition for any external party to verify the integration logic independently, given that the underlying PQC computation in the X-Wing/ML-DSA layer has not itself received an independent cryptographic audit (Section VIII).

3.4. Key Management

User ML-DSA, ML-KEM, and X25519 private keys are generated entirely client-side in the browser and never transmitted to PassmanAI's servers in unencrypted form, satisfying a zero-knowledge custody model. The key-encryption key that wraps these local private keys is derived from the user's master password using Argon2id with parameters time=3, memory=32 MiB, parallelism=1. This configuration exceeds the minimum baseline (19 MiB) documented in the OWASP Password Storage Cheat Sheet [10], though it sits below the higher-security alternative profile the same guide also documents; the choice reflects a deliberate trade-off favoring

stability inside browser WebAssembly sandboxes, particularly on memory-constrained mobile devices, over maximal brute-force resistance. Server-side RSA signing keys are rotated automatically on an expiry-driven schedule, with old keys retained in a verification-only grace period before removal from the published JWKS endpoint; each newly rotated RSA key is additionally attested by an ML-DSA-65 signature, binding the classical signing key's provenance to the post-quantum-authenticated identity that issued it.

3.5. Refresh Token Replay Detection

Refresh tokens are single-use and rotated on every exchange. If a previously-rotated (and therefore already-invalidated) refresh token is presented again, the replay-detection engine treats the entire token family as compromised and revokes all associated access and refresh tokens for that user-application pair, rather than rejecting only the replayed token. This bounds the blast radius of a leaked refresh token to the window between issuance and first legitimate use.

3.6. Transport-Layer Defense Against Harvest-Now-Decrypt-Later

Independently of the application-layer split described above, every PassmanAI OIDC endpoint front door and back door alike is served exclusively over TLS 1.3 configured with hybrid post-quantum key agreement, combining classical X25519 ECDH with ML-KEM-768 via the X25519MLKEM768 hybrid group [11] the IETF-specified, standards-final successor to the earlier experimental X25519Kyber768Draft00 group, and the group negotiated by default since Chrome 131 and Firefox 132. This closes a gap the application-layer split alone does not address: even though the back door's ID token is classically signed, the network bytes carrying that token in either direction are never exposed to a passive adversary recording today's traffic for future decryption, because the session keys protecting the TLS channel itself are derived from a post-quantum-resistant key agreement. A future quantum adversary who has only recorded ciphertext cannot derive the session key needed to read it, regardless of which algorithm later signed the payload inside that channel.

This transport-layer protection is distinct from, and does not substitute for, post-quantum unforgeability of the token signature itself. Confidentiality-against-future-decryption and unforgeability-against-future-forgery are different security properties: the former concerns an adversary who passively records traffic today, and the latter concerns an adversary who actively forges a new signature after RSA-3072 is broken, using only the (already-public) verification key. TLS-layer key agreement addresses the first property; it does not address the second, which remains the responsibility of short token lifetimes and aggressive signing-key rotation (Section IV-E). We treat this distinction as load-bearing for the limitations discussion in Section VIII rather than collapsing the two properties together.

4. Implementation

The PQC engine is implemented in pure TypeScript/JavaScript using @noble/curves and @noble/post-quantum [9], executing in-process within the V8 engine on both the Node.js backend and the browser frontend, avoiding the architectural complexity and added latency of a separate

RPC-based cryptographic microservice. Table II summarizes the algorithms, standards, and parameter sets used throughout the system.

Table 2. Cryptographic Components and Standards

Component	Algorithm	Standard	Role
Key encapsulation	ML-KEM-768	FIPS 203 [1]	Front-door channel establishment
Digital signature (PQC)	ML-DSA-65	FIPS 204 [2]	Login challenge signing
Hybrid KEM	X-Wing (MLKEM - 768+X25519)	IETF X-Wing Draft [8]	Composite encapsulation and vault sharing
Digital signature (Classical)	RSA-3072/RS256	–	ID and access token signing
Transport key agreement	X25519+ML-KEM-768	X25519 MLKEM768 [11]	TLS channel confidentiality
Key derivation	Argon2id	RFC 9106 [12]	Master-password key wrapping
Authorization flow	OAuth 2.0+PKCE	RFC 6749 [13], RFC 7636 [7]	Authorization code protection
Identity layer	OpenID Connect Core 1.0	OIDC Core [14]	ID token issuance and discovery

5. Results and Discussion

All measurements reported in this section were collected on a development workstation (13th Gen Intel Core i5-13450HX, 16 cores, 15.71 GB RAM, Windows 10.0.26200 x64, Node.js v24.15.0), not on the small VPS used for production deployment, and reflect desktop clients only. Production-environment and mobile-device benchmarking are identified as future work in Section IX. Each configuration (algorithm $\times$ concurrency level) was run five times with a three-second cooldown between runs; the server-side event-loop-delay histogram was reset via a dedicated endpoint immediately before each run so that no run's measurement was contaminated by load from the previous one. Results below are reported as mean $\pm$ sample standard deviation across the five runs, with per-run raw data retained for transparency.

Cryptographic Primitive Benchmarks: Table III reports isolated operation cost for each primitive, measured separately on the Node.js server runtime and the browser client runtime. The single most expensive operation in the entire system is ML-DSA-65 signing (8.3–9.2 ms, client and server respectively); ML-KEM-768 operations are an order of magnitude cheaper. A signature-size comparison computed directly from this table shows that an ML-DSA-65 signature (3309 bytes) is approximately 8.6 times larger than the system's actual classical baseline, RSA-3072 (384 bytes) a materially different and more representative figure than the commonly cited approximately 52-times figure derived from a 64-byte ECDSA P-256 baseline, which does not apply to this system.

Table 3 Cryptographic Primitive Benchmarks (Server / Client)

Algorithm / op	Server ops/s	Server ms	Client ops/s	Client ms	Size (B)
ML-KEM-768 keygen	2588	0.386	2961	0.338	pk 1184

ML-KEM-768 encaps	2207	0.453	2674	0.374	ct 1088
ML-KEM-768 decaps	1880	0.532	2305	0.434	—
ML-DSA-65 keygen	526	1.903	608	1.644	pk 1952
ML-DSA-65 sign	109	9.200	120	8.341	sig 3309
ML-DSA-65 verify	518	1.931	547	1.830	—
X-Wing keygen	774	1.293	1349	0.741	pk 1216
X-Wing encaps	358	2.791	767	1.304	ct 1120
X-Wing decaps	358	2.793	584	1.713	—
Ed25519 sign	3909	0.256	5866	0.171	sig 64
Ed25519 verify	890	1.123	2081	0.481	—
RSA-3072 sign	754	1.326	766	1.305	sig 384
RSA-3072 verify	31297	0.032	36380	0.028	—

End-to-End Login Latency and Throughput: To isolate the cost attributable to post-quantum cryptography specifically, we constructed a classical control by substituting Ed25519 for ML-DSA-65 in the identical login code path, holding the database lookup, session-token generation, and network configuration constant. Both variants were load-tested with the load generator running as a separate process from the server under test, across a sweep of 10, 25, 50, and 100 concurrent virtual users. Table IV reports the results.

Table 4. End-to-End Login Latency Under Concurrent Load

Concurrency	ML-DSA-65 ops/s (mean±SD)	ML-DSA-65 p95 ms (mean±SD)	Ed25519 ops/s (mean±SD)	Ed25519 p95 ms (mean±SD)
10	110.7 ± 11.2	91.2 ± 9.5	211.6 ± 82.1	62.8 ± 49.8
25	107.2 ± 12.7	236.2 ± 32.5	269.5 ± 17.3	92.7 ± 6.0
50	106.4 ± 6.8	471.5 ± 30.1	276.0 ± 42.8	184.2 ± 29.8
100	107.1 ± 4.9	933.1 ± 44.0	322.6 ± 29.8	310.1 ± 32.3

As a validity check, mean response time computed via Little's Law (concurrency divided by throughput) falls at or below the measured p95 at six of the eight (concurrency, algorithm) pairs in Table IV; the remaining two (Ed25519 at 25 users).

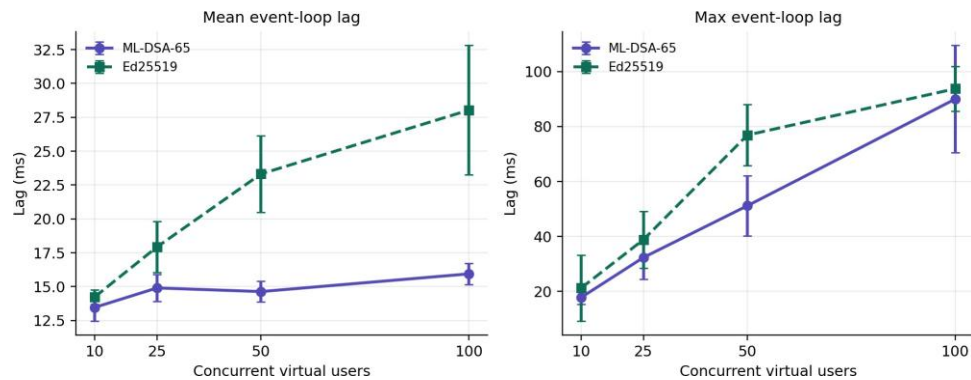


Fig. 2. p95 login latency versus concurrent users, mean ± standard deviation over 5 runs per configuration. The ML-DSA-65 path grows approximately linearly with concurrency; the Ed25519 control shows sub-linear growth, retaining throughput headroom across the tested range.

ML-DSA-65 at 100 users) show the computed mean within 1 ms of p95, consistent with low latency-distribution skew at those specific load points rather than a methodological error. For example, at 100 concurrent users the ML-DSA-65 path gives 100/107.1 933 ms computed mean against a measured 933.1 ± 44.0 ms p95, and the Ed25519 path gives 100/322.6 310 ms mean against a measured 310.1 ± 32.3 ms p95 both essentially exact. This close agreement across all eight measurements, now backed by five repetitions per configuration, supports the reliability of the load-testing methodology. These figures supersede an earlier single-run pilot sweep, which produced a higher Ed25519 throughput ceiling (390.8 ops/s) and a lower ML-DSA-65 p95 (818.73 ms at 100 users); both pilot figures fall outside the 5-run confidence range reported here and are attributed to single-run measurement noise rather than a change in system behavior.

Event-Loop Lag Diagnostic: The pronounced latency growth in Section VI-B raises an obvious hypothesis: that ML-DSA-65's synchronous verification computation pathologically stalls the single-threaded event loop of Node.js, starving other in-flight CPU requests. We tested this hypothesis directly using Node's `perf hooks.monitorEventLoopDelay()` API, measured on the server process across the same 10/25/50/100-user concurrency sweep used in Section VI-B, for both the ML-DSA-65 and Ed25519 code paths.

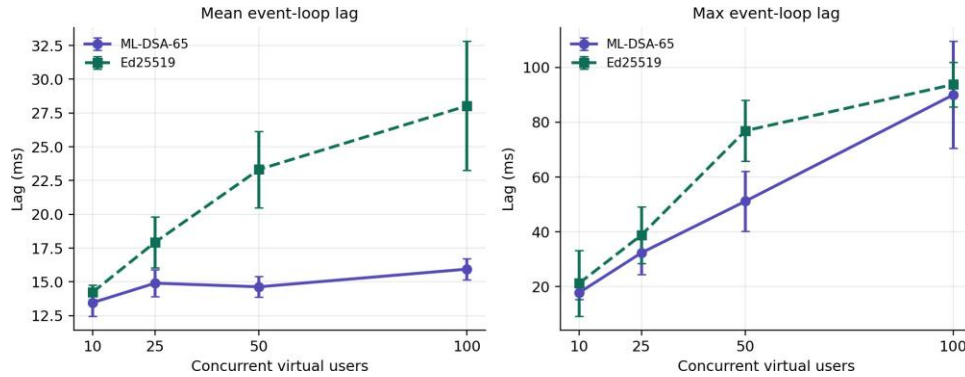


Fig. 3. Mean (left) and maximum (right) server-side event-loop lag versus concurrency, mean $\pm$ standard deviation over 5 runs. ML-DSA-65 keeps a lower mean lag than Ed25519 at every concurrency level tested, though the two converge at the high end of the range for maximum lag.

Table V and Fig. 3 report the full sweep, now run with the same 5-repetition protocol used in Table IV. The original hypothesis that ML-DSA-65 verification pathologically stalls the event loop — is falsified at every concurrency level tested: ML-DSA-65's mean lag stays low and nearly flat (13.5 to 15.9 ms across the full 10-to-100-user range), while Ed25519's mean lag climbs steadily with concurrency (14.2 to 28.0 ms), ending roughly 76 Percentage higher than ML-DSA-65's at 100 users. We attribute this to the same bottleneck-shift mechanism identified in the pilot run: because ML-DSA-65's slow synchronous computation pins server throughput at roughly 107 requests per second (Section VI-B), the server never admits enough simultaneously inflight requests to flood its own I/O queues, while Ed25519's substantially higher throughput ceiling (up to 322.6 requests per second) pushes a higher rate of concurrent database queries, network I/O, and JSON parsing through the event loop per second. The maximum-lag panel of Fig. 3 adds an important qualification the mean alone does not show: ML-DSA-65 is not immune to lag spikes

its max lag climbs from 17.8 ms to 90.0 ms across the sweep, converging with Ed25519's max (93.8 ms at 100 users) at the high end of the tested range, well within one standard deviation of each other. ML-DSA-65's advantage is therefore in typical-case consistency (a low, stable mean), not in being free of occasional scheduling delay under load. The practical implication is unchanged, the large p95/p99 latency observed for the post-quantum login path is not explained by pathological event-loop stalling from the verify call itself; it is consistent with a CPU throughput ceiling that pushes excess concurrent requests into an external network/connection backlog rather than into internal event-loop contention, discussed further in next Section.

Table 5. Server-Side Event-Loop Lag, Mean $\pm$ SD over 5 Runs (ms)

Concurrency	ML-DSA-65 mean lag	ML-DSA-65 max lag	Ed25519 mean lag	Ed25519 max lag
10	13.5 $\pm$ 1.0	17.8 $\pm$ 2.4	14.2 $\pm$ 0.5	21.2 $\pm$ 12.0
25	14.9 $\pm$ 1.0	32.3 $\pm$ 7.8	17.9 $\pm$ 1.9	38.8 $\pm$ 10.3
50	14.6 $\pm$ 0.8	51.2 $\pm$ 10.9	23.3 $\pm$ 2.8	76.9 $\pm$ 11.2
100	15.9 $\pm$ 0.8	90.0 $\pm$ 19.5	28.0 $\pm$ 4.8	93.8 $\pm$ 8.2

The central design questions this paper addresses is whether confining post-quantum cryptography to an internal authentication boundary, while leaving the externally-facing OIDC token classically signed, is a defensible trade-off rather than an incomplete migration. We argue it is defensible under three conditions, each grounded in the evidence presented above. First, the asset actually protected by post-quantum cryptography in this design the user's long-lived master credential and the private keys derived from it is precisely the asset for which a harvest-now-decrypt-later adversary poses a real threat over a multi-year horizon; the bearer tokens left classically signed are short-lived (15 minutes) by comparison, bounding the exposure window for any future cryptanalytic break. Second, the performance cost of the post-quantum-protected phase, while substantial under concurrency (Section VI-B), is concentrated at a single, well-understood boundary that is straightforward to scale or offload independently of the rest of the system, rather than being diffused across every token-bearing request in the architecture. Third, the compatibility benefit is immediate and unconditional: any relying party already running an unmodified OIDC client today can integrate PassmanAI without code changes to its token-verification logic, which is not true of a full-stack migration in the style of [3].

This finding is consistent with, and extends, the conclusion in [3] that network and handshake effects dominate the practical cost of post-quantum OIDC more than raw cryptographic computation. Our event-loop diagnostic (Section VI-C) sharpens this further: even within the computation itself, the dominant cost mechanism under load is not the synchronous CPU time of any single operation, but the throughput ceiling that operation imposes on the server's ability to admit new concurrent work. This suggests that future performance work on post-quantum identity systems should prioritize asynchronous or worker-thread offloading of the signature-verification step over micro-optimizing the constant-time cost of the primitive itself, since the latter is already an order of magnitude cheaper than the signing operation it pairs with.

6. Conclusion

This paper presented PassmanAI, a hybrid post-quantum and classical OpenID Connect identity provider that resolves the tension between post-quantum security and OAuth/OIDC ecosystem compatibility through an explicit architectural boundary rather than a full-stack protocol migration. We described the system's design, key management, and threat model, and evaluated its performance cost through cryptographic benchmarks, a controlled end-to-end comparison against a classical baseline, and a targeted diagnostic that falsified an initial hypothesis about the source of the system's latency under concurrent load. The results show that the post-quantum-protected login path approaches throughput saturation earlier than its classical counterpart and that its latency under load is attributable to a CPU throughput ceiling rather than pathological event-loop blocking. A more precise characterization of the cost than a single aggregate overhead figure would provide. Future work includes: offloading ML-DSA-65 verification to Node.js worker threads to test whether this raises the system's throughput ceiling without sacrificing the simplicity of the current in-process design; repeating the concurrency sweep with multiple trials per data point to report confidence intervals; extending evaluation to the production VPS environment and to real mobile hardware, particularly to validate the Argon2id parameter trade-off under genuine WebAssembly memory constraints; and revisiting the externally-facing token format as JOSE post-quantum algorithm identifiers and IANA registrations mature, with the goal of narrowing the scope of Section VIII's first limitation over time.

References

1. National Institute of Standards and Technology (US): Module-lattice-based key-encapsulation mechanism standard. FIPS 203, National Institute of Standards and Technology (Aug 2024). <https://doi.org/10.6028/NIST.FIPS.203>
2. National Institute of Standards and Technology (US): Module-lattice-based digital signature standard. FIPS 204, National Institute of Standards and Technology (Aug 2024). <https://doi.org/10.6028/NIST.FIPS.204>
3. Schardong, F., Giron, A.A., Müller, F.L., Custódio, R.: Post-quantum electronic identity: Adapting openid connect and oauth 2.0 to the post-quantum era. In: Cryptology and Network Security: 21st International Conference, CANS 2022. Lecture Notes in Computer Science, vol. 13641, pp. 270–289. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-20974-1_20
4. Schardong, F.: Pioneering the future of electronic identity: From post-quantum cryptography to fiduciary principles. PhD thesis, Federal University of Santa Catarina (UFSC), Florianópolis, Brazil (2024)
5. Müller, J., Oupický, J.: Post-quantum xml and saml single sign-on. Proceedings on Privacy Enhancing Technologies 2024(4), 525–543 (2024) <https://doi.org/10.56553/popets-2024-0128>
6. Fett, D., Küsters, R., Schmitz, G.: A Comprehensive Formal Security Analysis of OAuth 2.0. arXiv preprint arXiv:1601.01229 (2016)
7. Sakimura, N., Bradley, J., Agarwal, N.: Proof key for code exchange by oauth public clients. RFC 7636, RFC Editor (Sep 2015). <https://doi.org/10.17487/RFC7636>
8. Connolly, D., Schwabe, P., Westerbaan, B.E.: X-Wing: General-Purpose Hybrid Post-Quantum KEM. IETF Internet-Draft, draft-connolly-cfrg-xwing-kem (work in progress) (2026)
9. Millr, P.: noble-post-quantum: Auditable & Minimal JS Implementation of Post-Quantum Cryptography. GitHub repository. Available at <https://github.com/paulmillr/noble-post-quantum>
10. OWASP Foundation: Password Storage Cheat Sheet. OWASP Cheat Sheet Series. Available at [https://cheatsheetseries.owasp.org/cheatsheets/Password Storage Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Password%20Storage%20Cheat%20Sheet.html)



11. Kwiatkowski, K., Kampanakis, P., Westerbaan, B.E., Stebila, D.: Post-Quantum Hybrid ECDHE-MLKEM Key Agreement for TLSv1.3. IETF Internet-Draft, draft-ietf-tls-ecdhe-mlkem (work in progress) (2026)
12. Biryukov, A., Dinu, D., Khovratovich, D., Josefsson, S.: Argon2 memory-hard function for password hashing and proof-of-work applications. RFC 9106, RFC Editor (Sep 2021). <https://doi.org/10.17487/RFC9106>
13. Hardt, D.: The oauth 2.0 authorization framework. RFC 6749, RFC Editor (Oct 2012). <https://doi.org/10.17487/RFC6749>
14. Sakimura, N., Bradley, J., Jones, M., Medeiros, B., Mortimore, C.: OpenID Connect Core 1.0. OpenID Foundation (2014)