qID: A Post-Quantum Identity Construction from NIST-Standardized Primitives

Mende Matthias · the qID Project · qid.dev

July 2026

CONTENTS

qID: A Post-Quantum Identity Construction from NIST-Standardized Primitives

Mende Matthias · the qID Project · qid.dev Technical Report, v0.3.0 · July 2026 · *Preprint, pending independent audit*

> *"Self-custody and self-sovereignty are human rights. In the quantum age, defending them is an engineering duty, not a feature."* — Mende Matthias

Abstract

We present **qID**, a self-custodial identity construction in which a single 32-byte seed deterministically derives a complete post-quantum identity: an on-chain settlement address, transaction signing keys, an authentication ("login") capability, a cold recovery capability, and public-key encryption. qID introduces **no new cryptography**. It is an engineering composition of the three NIST post-quantum standards finalized in August 2024, namely ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205), arranged in a two-layer architecture. A *cold, hash-based root* (SLH-DSA-128s) anchors the identity under the most conservative cryptographic assumption available, while *hot, lattice-based operational keys* (ML-DSA-44) carry the daily signing load and remain rotatable and revocable under that root. The construction is implemented in under 1,000 lines of dependency-pinned JavaScript, is byte-exact to the BTX blockchain's consensus rules (verified against a live node via known-answer address derivation and `testmempoolaccept`), and is deployed today in three independent wallet integrations. We specify canonical byte layouts for every off-chain signed object, publish language-neutral conformance vectors, and describe a verification methodology built from NIST ACVP known-answer tests, differential cross-verification against the chain's own C++ signer, property-based fuzzing, and *negative-space tests* that fail if hedged randomness or transaction commitments are ever silently degraded. We report the measured costs of post-quantum self-custody: a 2-input/2-output transfer occupies 7,734 vbytes, roughly 37 times its ECDSA equivalent, and hash-based cold-key generation, pathologically slow in interpreted JavaScript on mobile at roughly 243 seconds, reaches roughly 233 milliseconds when routed through a platform crypto engine. That is a thousandfold improvement with byte-identical output. We close with an honest security analysis, including what each key's compromise yields, the side-channel residuals we accept, and the limits of the seed-derived recovery model.

---

1. Introduction

1.1 MOTIVATION

Virtually all deployed blockchain and Web3 identity today rests on elliptic-curve cryptography (ECDSA/secp256k1, Ed25519), which is broken in polynomial time by a cryptographically relevant quantum computer running Shor's algorithm [1]. The threat is not confined to the future. Any public key visible on-chain today can be recorded now and attacked later, the pattern known as "harvest now, decrypt later" [2]. This motivated NIST's post-quantum standardization program, which concluded its first round in August 2024 with three standards: FIPS 203 (ML-KEM, key encapsulation, derived from CRYSTALS-Kyber [5]), FIPS 204 (ML-DSA, signatures, derived from CRYSTALS-Dilithium [4]), and FIPS 205 (SLH-DSA, stateless hash-based signatures, derived from SPHINCS+ [6]) [3].

Standards, however, are primitives, not systems. An *identity* additionally requires answers to lifecycle questions the primitives do not address. How does one seed become many keys, deterministically and portably? What happens when the everyday signing key leaks? How does a relying party learn that a key was rotated away? How is an encryption key bound to an identity so it cannot be substituted in transit? qID is a concrete, deployed answer to these questions, built exclusively from standardized components.

1.2 DESIGN GOALS

1. **One seed, total custody.** Every capability derives from one 32-byte secret the user holds.
2. **Conservative anchor.** The *identity* must not rest on a single mathematical assumption. The stable root is hash-based (SLH-DSA), whose security reduces to the preimage resistance of SHA-3/SHAKE, the most scrutinized assumption in the portfolio.
3. **Hot/cold asymmetry.** Fast lattice signatures do the daily work; the hash-based root is touched only for rotation, revocation, and last-resort fund recovery.
4. **Byte-exactness.** Where qID meets a blockchain, concretely **BTX**, a Bitcoin-derived UTXO chain whose consensus is natively post-quantum (every output is a P2MR script, and ECDSA is not consensus-valid on its L1), it reproduces the chain's consensus bytes exactly, verified against the chain's own node software rather than a reimplementation.
5. **Reimplementability.** Everything signed is signed over an explicit, versioned byte layout, and conformance vectors allow an independent implementation to prove agreement byte for byte.
6. **Honesty.** No "unhackable" claims. Residual risks are documented, and the construction is labeled *pending independent audit*.

1.3 NON-GOALS AND NON-CLAIMS

qID does not propose new cryptographic schemes, does not claim security proofs beyond those of its standardized components, and does not claim quantum safety for the optional classical components it can derive for interoperability (a secp256k1 EVM key, disabled by default, and the X25519 half of hybrid encryption, which only ever *adds* to security; see §6.3).

2. Preliminaries

2.1 PRIMITIVES AND PARAMETERS

qID fixes one parameter set per role. Table 1 lists the standardized primitives with their NIST security categories and concrete sizes as used in qID.

Table 1. Primitives used by qID.

Role	Scheme	Standard	NIST cat.	Public key	Signature / CT	Secret key
Hot signing (login + spend)	ML-DSA-44	FIPS 204 [3]	2	1,312 B	2,420 B	2,560 B
Cold root (identity, recovery)	SLH-DSA-SHAKE-128s	FIPS 205 [3]	1	32 B	7,856 B	64 B
Encryption (KEM)	ML-KEM-768	FIPS 203 [3]	3	1,184 B	1,088 B (ct)	2,400 B
Hybrid DH (additive)	X25519	RFC 7748 [8]	n/a (classical)	32 B	32 B (epk)	32 B
Hashing / KDF	SHA-256, HKDF	FIPS 180-4, RFC 5869 [7]	n/a	n/a	n/a	n/a

The asymmetry between the two signature schemes is the design's engine. SLH-DSA-128s has a *32-byte* public key (it fits in a Merkle leaf and in a QR code) but large signatures and slow key generation; ML-DSA-44 signs in milliseconds with moderate sizes. qID therefore puts SLH-DSA where signatures are rare (rotation, revocation, rescue) and ML-DSA where they are frequent (every login, every spend).

2.2 TAGGED HASHING

All qID domain separation uses BIP-340-style tagged hashes [11]:

```
H_tag(m) = SHA-256( SHA-256(tag) || SHA-256(tag) || m )
```

Distinct tags (`"P2MRLeaf"`, `"P2MRBranch"`, `"TapSighash"`, `"BTX-qID/login-v2"`, `"BTX-qID/attest-v3"`, `"BTX-qID/revoke-v1"`) render cross-protocol signature reuse structurally meaningless: a signature over one domain cannot verify in another.

3. Deterministic derivation: one seed, many keys

3.1 THE DERIVATION CHAIN

Let `s ∈ {0,1}^256` be the master seed. For a key of algorithm `a ∈ {ML-DSA=0x00, SLH-DSA=0x01}` at BIP-32-style coordinates (change `c`, index `i`), qID derives (byte-exact to the BTX node's `pq_keyderivation.cpp`):

```
info  = "m/87h" || BE32(87') || BE32(coin') || BE32(account') || BE32(c) || BE32(i) || a
sm    = HKDF-SHA256(ikm = s, salt = "BTX-PQ-BIP87-HKDF-V1", info, L = 32)      (Eq. 1)
E     = SHA-256(sm || LE32(0)) || SHA-256(sm || LE32(1)) ||
        SHA-256(sm || LE32(2)) || SHA-256(sm || LE32(3))                        (Eq. 2)
```

`E` is 128 bytes of stretched entropy. FIPS keygen then consumes exactly its specified seed material: ML-DSA-44 takes `ξ = E[0:32]`, and SLH-DSA-128s takes `E[0:48]` as `SK.seed || SK.prf || PK.seed`. The apostrophes denote hardened components (`x' = x + 2³¹`). ML-KEM-768 derives analogously on a separate

purpose branch (m/88') with its own salt, taking 64 bytes as `d || z`. Two qID-owned branches (distinct HKDF salts `"qID-AUTH-MLDSA-V1"` and `"qID-X25519-V1"`) derive an optional auth-only ML-DSA key that never appears in any spend script, and the X25519 hybrid key.

Because HKDF accepts arbitrary-length input keying material silently, qID rejects seeds outside BIP-32's accepted range of 16 to 64 bytes at every entry point. A truncated seed would otherwise derive a weaker but *valid-looking, fundable* identity, a failure mode we class as worse than an exception.

3.2 THE IDENTITY AS A MERKLE COMMITMENT (P2MR)

The on-chain identity is a **pay-to-Merkle-root** (P2MR) output: a 32-byte root committing to two script leaves, one per signature scheme:

```
leaf_login    = push(pk_MLDSA)  || OP_CHECKSIG_MLDSA   (0xbb)
leaf_recovery = push(pk_SLHDSA) || OP_CHECKSIG_SLHDSA  (0xbc)

h_i   = H_"P2MRLeaf"( 0xc2 || compactSize(|leaf_i|) || leaf_i )
root  = H_"P2MRBranch"( min(h_0, h_1) || max(h_0, h_1) )                    (Eq. 3)
addr  = bech32m("btx", v=2 || root)          [BIP-350 encoding, 12]
```

Spending reveals only the executed leaf plus a 33-byte control block (leaf version 0xc2 and the sibling hash); the *unused* leaf stays a hash. In the common case the SLH-DSA public key never appears on-chain at all. The address is thus three things at once: a spendable output, a commitment to the recovery policy, and a rotatable handle, since a new login key yields a new root while the identity (§5) persists.

Every derivation in Eqs. 1 through 3 was verified byte-identical against the BTX node: the node's `deriveaddresses` on the canonical descriptor `mr(pqhd(...), pk_slh(pqhd(...)))` returns exactly the address computed by qID for the standard test seed, and this equality is pinned as an offline known-answer test. Figure 1 summarizes the full derivation tree.

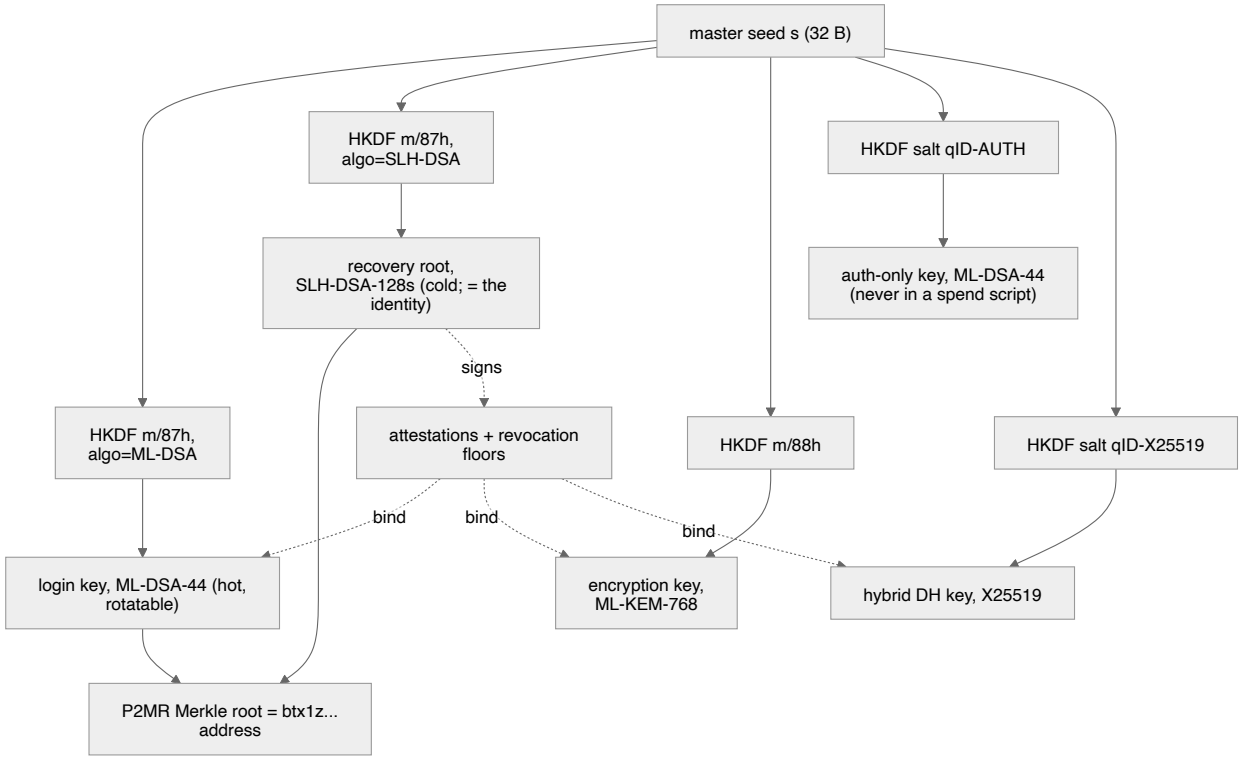

Figure 1. One seed, many keys: the qID derivation tree. Solid arrows are key derivation; dotted arrows are signatures/bindings.

4. The on-chain layer

4.1 TRANSACTION COMMITMENT

P2MR spends sign a BIP-341-style sighash [11] (epoch 0x02) whose preimage commits to *all* prevouts, amounts, scriptPubKeys, sequences and outputs via precomputed midstate hashes, plus the executing input's index and leaf hash. Consequently a data provider (for example, a block explorer) that lies about an input's amount cannot redirect value: the signature simply fails node-side. qID's builders additionally range-check every amount against the fixed supply (2.1×10^{15} satoshi) *before* serialization, because the int64 serializer would otherwise wrap silently.

4.2 THE MEASURED PRICE OF POST-QUANTUM SELF-CUSTODY

Post-quantum witnesses are large, and BTX grants them no witness discount, so `vsize` equals raw size. Measured node-exact (regtest, `testmempoolaccept`):

Table 2. Measured transaction sizes (1 P2MR output unless noted).

Inputs	vsize (vB)	Per-input marginal
1	3,873	n/a
2	7,691	+3,818
3	11,509	+3,818
2 (2 outputs)	7,734	n/a

Each ML-DSA login-leaf input costs a fixed 3,818 vB (41 B non-witness plus 3,777 B witness: 2,420 B signature, 1,316 B leaf script, 33 B control block, plus length prefixes). For comparison, a 2-in/2-out P2WPKH ECDSA transaction is about 209 vB, so post-quantum self-custody on this design costs about **37 times** the block space of its classical equivalent. We consider that the honest, and in our view acceptable, price of removing the discrete-logarithm assumption from custody. Fee estimation is therefore exact rather than heuristic: `fee = vsize(n_in, outputs) × rate`, with a 1 sat/vB relay floor and a hard max-fee clamp enforced *after* dust folding, so the clamp is a true ceiling.

4.3 CONSENSUS MIGRATION IN PRODUCTION: THE C-002 EVENT

qID's original interoperability finding was that the chain advertised "SLH-DSA (FIPS 205)" while consensus-accepting only round-3.1 SPHINCS+ signatures (no FIPS-205 domain prefix, pre-standard FORS indexing), so standards-compliant signers could not produce recovery spends. The core team resolved this with a height-gated consensus migration ("C-002"):

```
pre-activation:   verify_r3.1( M, σ )
post-activation:  verify_r3.1-core( M' , σ )  with  M' = 0x00 || 0x00 || M     (Eq. 4)
                  plus FIPS-205 FORS message-to-index extraction
```

`M' = toByte(0,1) || toByte(|ctx|=0,1) || M` is precisely FIPS 205's pure-mode, empty-context message wrapper (the `slh_sign` / `slh_verify` interface). We verified the equivalence *empirically and bidirectionally* by compiling the chain's own C signer at the migration commit and cross-verifying against an independent FIPS-205 implementation [13]:

Table 3. SLH-DSA cross-verification matrix (chain core at commit `f3c9eb77fa` vs. noble).

#	Direction	Result
1	FIPS-205 sign(M) → core verify(M′, fips205=1)	**VALID**
2	core sign(M′, fips205=1) → FIPS-205 verify(M)	**VALID**
3	FIPS-205 sign(M) → core legacy verify(M, 0)	INVALID (expected)
4	core legacy sign → core legacy verify	VALID
5	legacy σ → fips205 verify	INVALID (clean domain separation)

Rows 1 and 2 establish two-way equivalence; row 5 shows no cross-era signature malleability. The migration activated at mainnet height 123,000 and, at the time of writing (tip 153,813, 90-second target spacing), has been consensus-active for about 31,000 blocks, roughly 32 days. Any FIPS-205-conformant signer, including future hardware, can now produce recovery spends.

5. The off-chain identity layer

5.1 ATTESTATION: THE COLD ROOT VOUCHES FOR HOT KEYS

The stable identity is the SLH-DSA public key (32 bytes). Hot keys are bound to it by a CA-style **attestation** signed by the root, over a canonical byte layout (v3):

```
payload = 0x03 || root(32) || pk_login(1312) || pk_kem(1184)
        || (0x01 || pk_x25519(32) | 0x00)
        || i64LE(valid_from) || i64LE(valid_until) || i64LE(serial)
        || compactSize(n_caps) || Π_i ( compactSize(|cap_i|) || cap_i )
σ = SLH-DSA.Sign( sk_root , H_"BTX-qID/attest-v3"(payload) )                    (Eq. 5)
```

Nothing signed is ever JSON. Two independent implementations either agree on these bytes or fail loudly, which we consider a prerequisite for calling anything a standard. Capabilities (`caps ⊆ {login, spend, encrypt}`) are signed *and enforced*: a verifier refuses a login under an encrypt-only attestation. The default validity window is 30 days. Attestations are cheap and reissued on use, so a leaked but unrotated hot key has a bounded blast radius.

5.2 LOGIN, ROTATION, REVOCATION

Authentication is challenge and response, WebAuthn-shaped. The relying party (RP) mints `(nonce_256, origin, ts)` ; the client signs the tagged hash of a canonical encoding (v2) with the login (or dedicated auth-only) key. The RP enforces issuance binding (it only accepts nonces it minted), single use, a two-sided freshness window, origin equality, attestation validity, capability, and serial monotonicity. Signers additionally refuse challenges whose origin differs from the page they run on. Without this *client-side* check, origin binding is decorative against phishing relay; the property WebAuthn obtains from the browser, a JS SDK must obtain from itself.

Rotation issues a fresh attestation with `serial + 1` under the unchanged root. RPs pin the highest serial seen *and the winning key at that serial*, so two keys cannot share a serial. Because rotation only protects RPs that observe it, qID adds **revocation floors**: a root-signed statement (canonical layout, tag `"BTX-qID/revoke-v1"`) that `serial < n` is invalid *forever*, ingestible from any channel. The signature, not the transport, carries the trust. Floors ratchet monotonically upward.

5.3 HYBRID ENCRYPTION

Encryption to an identity uses a KEM-DEM with an *additive* classical component, following the deployed practice of TLS X25519MLKEM768 [9] and Signal PQXDH [10]:

```
(ct, ss_pq)  = ML-KEM-768.Encaps(pk_kem)
esk ←$ {0,1}^256 ;  epk = X25519.Pub(esk) ;  ss_dh = X25519(esk, pk_x25519)
K    = HKDF-SHA256( ss_pq || ss_dh , salt="qID/kem-hybrid-v2", info="aead-key" )
C    = AES-256-GCM_K( m ; AAD = ct || epk )                                     (Eq. 6)
```

Confidentiality holds if *either* ML-KEM-768 or X25519 holds. Both public transcripts are bound as AEAD associated data, so swapping either component fails authentication; all-zero X25519 outputs (low-order points) are rejected on both sides. The recipient keys are attested (§5.1), so a substituted encryption key fails verification before any ciphertext is produced, which addresses the classic unauthenticated-KEM substitution attack. Neither mode provides forward secrecy (recipient keys are long-lived); ratcheting is future work (§8).

6. Implementation and verification methodology

6.1 IMPLEMENTATION

The reference implementation is 996 lines of JavaScript (an 840-line core plus a 156-line relying-party verifier) over the audited noble/scure cryptographic libraries [13], pinned to exact versions and bundled; the shipped artifact makes zero network requests and contains zero `Math.random` call sites (enforced by test). Consumers vendor the bundle **byte-frozen** and hash-pinned. CI proves the bundle rebuilds byte-identically from the pinned source and publishes the (SHA-256 ↔ commit) provenance pair, upgrading the pin from "same bytes as before" to "bytes of *this* reviewable source."

6.2 FIVE VERIFICATION LAYERS

1. **Byte-exactness KATs.** Node-cross-checked vectors pin every derived key, address, and sighash; any drift in derivation or serialization fails offline, with no node required.
2. **Standards conformance.** NIST ACVP vectors anchor the FIPS-203/204/205 implementations to the standards themselves, not to their own history.
3. **Differential testing.** The chain's own C++ signer, compiled from the consensus source, is cross-verified against an independent implementation (Table 3).
4. **Property-based fuzzing.** Every attacker-facing parser and verifier holds a *total contract* (never throws, fails closed) under generated adversarial inputs; single-byte mutations of any ciphertext field must fail authentication.
5. **Negative-space tests.** Tests that assert the *presence of randomness and commitments* rather than correctness of output. Two ML-DSA signatures over the same message must differ (hedged signing is alive [3]), and mutating any input amount or output value by one satoshi must change every sighash (the value commitment is alive). These exist to make a silently backdoored rebuild fail loudly.

At v0.3.0 the suite comprises 73 tests and 993 assertions, all passing; wire formats are frozen.

6.3 PERFORMANCE ON CONSTRAINED TARGETS

ML-DSA-44 operations are fast everywhere (key generation about 130 ms, signing about 1 s on a mid-range phone's interpreted JS engine; milliseconds on desktop). SLH-DSA-128s key generation, however, is pathological in interpreted JavaScript on mobile: **about 243,000 ms** measured on a production React Native (Hermes) target. Routing exactly that operation through the platform's native crypto engine (OpenSSL `EVP_PKEY_keygen` with an injected FIPS-205 seed) reduced identity derivation to **about 233 ms, a roughly thousandfold improvement, with byte-identical output**, verified against the same NIST vectors as the reference path. We highlight this as the practical lesson of the deployment: post-quantum UX on mobile is an engineering problem with existing solutions, not a cryptographic blocker.

7. Security analysis

7.1 COMPROMISE SEMANTICS

Table 4. What each secret yields an attacker.

Compromised	Attacker gains	Recovery path
Login key (hot)	Spends from current address; logins until expiry/rotation	Root signs rotation + revocation floor; funds move to new root
Auth-only key	Logins only (never in any spend leaf)	Rotate; floor
KEM/X25519 keys	Decrypts ciphertexts to those keys (no forward secrecy)	Rotate encryption keys via new attestation
Recovery root (cold)	Full identity takeover	None; the root *is* the identity
Master seed	Everything above, forever	None

Two consequences are stated plainly rather than hidden. First, the login/recovery split defends against *hot-key* loss, not seed loss: both branches derive from one seed, and whoever holds it holds the identity. High-value deployments should hold the recovery role on a separately generated key (supported by the layered derivation, at the cost of the single-seed UX). Second, because a compromised *creation environment* poisons an identity from birth, qID ships a creation ceremony: re-derive from the re-entered seed on an independent device and compare addresses (`confirmIdentityAddress`), then fund a small test amount before real value.

7.2 SIDE CHANNELS

The single secret-dependent comparison is constant-time (XOR-accumulate). The lattice primitives themselves (rejection sampling, norm checks) have data-dependent timing inherent to the reference algorithms. On the deployed targets, with process-local keys and no remote timing oracle, we accept this residual explicitly rather than fork audited reference code, and document that shared-tenancy deployments would need to revisit it. ML-DSA signing is hedged (randomized) per FIPS 204's default, eliminating same-message nonce-style failure modes and verified alive by the negative-space suite; the SLH-DSA implementation signs deterministically, a variant FIPS 205 explicitly permits and one that carries no nonce-reuse risk by construction.

7.3 INPUT-SHAPE FAILURES

A recurring engineering theme: the dangerous failure is the one that *looks valid*. A truncated seed derives a fundable weak identity. Swapped public keys produce a valid-looking address whose leaves pair the wrong opcodes, leaving the funds unspendable. A "testnet" address decodes to the identical mainnet script because the human-readable prefix is display-only. qID therefore fails closed on all three: seed length gates, public-key shape gates on address construction, and network-strict address decoding (non-mainnet prefixes are opt-in for test tooling only).

7.4 LIMITATIONS

(i) No forward secrecy in the encryption layer. (ii) RP state (serials, nonces, floors) needs durable, shared storage in multi-instance deployments; the interface is pluggable but the reference store is single-process. (iii) The seed-derivation caveat of §7.1. (iv) SLH-DSA-128s targets NIST category 1; the design accepts this for the rarely-used root in exchange for the 32-byte public key, and category upgrades are a parameter change. (v) This construction has received adversarial internal review and the verification battery of §6, but **not yet an independent third-party audit**; that audit is scheduled before any real-value promotion beyond the current gated deployments.

8. Deployment status and future work

Three independent integrations vendor qID today, all pinning the same frozen bundle: a desktop self-custodial wallet (real mainnet value), a miner's sweep tool, and a consumer mobile app (feature-gated pending the audit). The C-002 migration (§4.3) is live on mainnet. Future work: ratcheted forward-secure messaging atop the hybrid KEM, M-of-N guardian recovery as additional Merkle leaves, durable RP store adapters, and growing the conformance-vector corpus as independent implementations appear.

9. Conclusion

qID demonstrates that a complete, honest post-quantum identity covering funds, login, recovery, encryption, rotation, and revocation can be assembled *today* from finalized NIST standards, with byte-level verifiability at every boundary and measured, acceptable costs. The construction's value is not novelty but composition discipline: conservative assumptions where permanence lives, fast assumptions where throughput lives, canonical bytes where interoperability lives, and tests that make even the *absence* of security properties loud.

References

[1] P. W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.* SIAM Journal on Computing 26(5), 1997.

[2] M. Mosca. *Cybersecurity in an Era with Quantum Computers: Will We Be Ready?* IEEE Security & Privacy 16(5), 2018.

[3] NIST. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*; *FIPS 204: Module-Lattice-Based Digital Signature Standard*; *FIPS 205: Stateless Hash-Based Digital Signature Standard.* U.S. Department of Commerce, August 2024. https://csrc.nist.gov/pubs/fips/203/finalSMARTPANTS_PRESERVED_613 · /204/final · /205/final

[4] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé. *CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme.* IACR TCHES 2018(1).

[5] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, D. Stehlé. *CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM.* IEEE EuroS&P 2018.

[6] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, P. Schwabe. *The SPHINCS+ Signature Framework.* ACM CCS 2019.

[7] H. Krawczyk, P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF).* RFC 5869, 2010.

[8] A. Langley, M. Hamburg, S. Turner. *Elliptic Curves for Security.* RFC 7748, 2016.

[9] IETF TLS Working Group. *Post-quantum hybrid ECDHE-MLKEM key agreement for TLSv1.3* (the X25519MLKEM768 group). Internet-Draft, work in progress, 2024.

[10] E. Kret, R. Schmidt. *The PQXDH Key Agreement Protocol.* Signal Foundation, 2023. https://signal.org/docs/specifications/pqxdh/SMARTPANTS_PRESERVED_643SMARTPANTS_PRESERVED_644

[11] P. Wuille, J. Nick, T. Ruffing / A. Towns. *BIP-340: Schnorr Signatures*; *BIP-341: Taproot: SegWit version 1 spending rules*. Bitcoin Improvement Proposals, 2020. https://github.com/bitcoin/bipsSMARTPANTS_PRESERVED_651SMARTPANTS_PRESERVED_652

[12] P. Wuille. *BIP-350: Bech32m format for v1+ witness addresses*. 2020. (Also BIP-32: *Hierarchical Deterministic Wallets*, 2012.)

[13] P. Miller. *noble-post-quantum, noble-hashes, noble-curves: auditable, minimal-dependency cryptography for JavaScript*. https://paulmillr.com/noble/SMARTPANTS_PRESERVED_663SMARTPANTS_PRESERVED_664

Artifacts. Live site: https://qid.devSMARTPANTS_PRESERVED_670 · Conformance vectors: https://qid.dev/vectors.jsonSMARTPANTS_PRESERVED_672 (mirrors `test/vectors.json`) · Specification: *BTX Post-Quantum Identity Convention*, v0.3, in the qID repository (opening with the independent audit) · Contact and security policy: SECURITY.md.

Acknowledgments. The BTX core team for the C-002 migration and review collaboration, and the bonuz engineering team for the mobile native-keygen integration and its measurements.