WP-036
June 2026
17 sections

POST-QUANTUM CRYPTOGRAPHY & LIVE VERIFICATION

Live PQC API Sandbox: Complete Post-Quantum Verification Field Report

We ran every panel at affix-io.com/sandbox, logged the JSON, and wrote it up. ML-DSA-65 attestations, Merkle inclusion on every call, offline edge tickets, 40+ Noir circuits, and latency numbers you can reproduce in fifteen minutes.

AffixIO Research | June 2026 | Download PDF | Open the sandbox

ABSTRACT

Post-quantum cryptography migration guides focus on TLS and PKI. They rarely give you a browser tab where you can mint a quantum-resistant audit record, verify it, and read the ML-DSA-65 signature in the response JSON. AffixIO's live API sandbox at affix-io.com/sandbox does exactly that. It proxies production ticket, QR, edge, and zero-knowledge endpoints; signs requests with `aio_web_demo`; and inclusion-checks every successful operation against the Merkle tree on `api.affix-io.com`. This paper is a full field report: architecture, API surface, response fields, circuit catalogue, latency benchmarks, privacy model, and a step-by-step reproduction checklist for security teams, auditors, and press. No mock Merkle roots. No server-side session

> store. Synthetic references only. Everything described here is observable in the public sandbox without credentials beyond opening the page.

CONTENTS

SECTION 01

Why we opened the sandbox

NIST published ML-KEM (FIPS 203) and ML-DSA (FIPS 204) in 2024. CNSA 2.0 timelines are public. Hybrid TLS with X25519MLKEM768 is shipping. Yet when a CISO asks "show me post-quantum verification working in our stack," the answer is usually a vendor slide or a curl against a staging server with fake data.

We wanted something falsifiable. AffixIO had published theory on stateless post-quantum verification, ML-DSA Merkle anchoring, and audit tree architecture. The sandbox claims live calls with Merkle inclusion on every success. We ran it to see if the claims hold under inspection.

They do. This document records a single session: eight audited operations across six circuit types, latencies from 77ms to 618ms, ML-DSA-65 attestations on identity and ZK verify paths, and a Merkle root that updates as the tree grows. You can repeat the same session without contacting AffixIO.

SECTION 02

What this is not

Clarity upfront saves time:

- **Not a mock.** Proxies hit live CMS and ZK infrastructure. The Merkle root is fetched from production, not generated in the browser.
- **Not a production API key.** Requests use the public `aio_web_demo` credential baked into the sandbox. Rate limits and quotas differ from paid integrations.
- **Not a PKI replacement demo.** You will not walk away with ML-DSA X.509 certificates. You will see ML-DSA-65 on attestations and Merkle anchors, which is the audit-layer complement to TLS and PKI migration described in WP-029.
- **Not storing your data.** Synthetic event names, fake holder labels, session-local storage only. Suitable for press, auditors, and security researchers without GDPR consent forms.

What it *is*: the fastest public path we know of to observe post-quantum cryptographic verification, zero-knowledge proof decisions, and Merkle audit inclusion in one sitting.

SECTION 03

Architecture: proxies, signing, Merkle

The sandbox UI is a thin client. All cryptography happens server-side. The browser calls CORS-enabled proxy routes that mirror integrator paths:

PROXY ROUTE	BACKEND	COVERS
`GET/POST /sandbox/api/cms/*`	CMS (port 3000)	Tickets, QR, edge issue/verify/consume
`GET/POST /sandbox/api/zk/*`	`api.affix-io.com`	Circuits, identity verify, Merkle audit tree

Every request carries the `aio_web_demo` signature. On page load, a health sweep checks CMS and ZK availability. Ours completed in 204ms.

The header bar shows three live indicators: session ID, API credential label, and the current Merkle root. Ours on load:

```
89c0225ccf7d1b9ea0a396327877fdd02037bd19b756b47e3d225b3f3d58aced
```

After operations, individual responses carry their own `merkle_validation.root` values reflecting the tree state at commit time. The global root updates when you hit Refresh status. That split matters for auditors: you can prove an operation was included at a specific index even if the global root has advanced since.

Audit path on success

On every successful mint, verify, redeem, or proof run, the backend:

1. Computes a `proofDigest` for the operation.
2. Appends a leaf to the Merkle audit tree with a `circuit_id` and event type (typically `verified`).
3. Returns `merkle_validation` with `valid: true`, root, leaf hash, and `inclusion_index`.

No separate "audit API call" is required. Inclusion is synchronous with the operation you already wanted to perform.

SECTION 04

Session model and privacy

The sandbox generates a session identifier (ours: `sb_mqmpeer2_31p15h`). State storage:

- **sessionStorage:** minted tokens, proof hex, activity log, panel form values.
- **Server:** no persistent sandbox session database described in the public UI.
- **Clear session** button wipes local state. Reload does the same.

For harvest-now-decrypt-later threat modelling, the relevant property is what never enters long-lived storage: holder identity, raw witness inputs, email, payment details. The Merkle tree records that a proof event occurred at a given circuit with a given digest. It does not need PII to function. That aligns with the stateless verification model in WP-030: post-quantum algorithms on the anchor, no harvestable identity database behind the verifier.

SECTION 05

The seven panels

PANEL	WHAT YOU TEST	PRIMARY PQC RELEVANCE
Overview	Proxy map, health sweep, circuit list, ticket schema	Orientation and API surface
Tickets	Mint, verify, redeem, revoke, status	Compact-v3 tokens, Merkle on ticket circuit
Edge	Issue, verify, consume, spent export	Offline verification, spent-proof anti-scalping
QR codes	URL and token QR generation	Proof-bound codes, qr_url circuit audit
ZK proofs	Identity verify, circuit prove/verify	ML-DSA-65 attestations, Noir engine
Merkle	Root display, inclusion context	Live audit tree inspection

PANEL	WHAT YOU TEST	PRIMARY PQC RELEVANCE
Activity	Request log with latency	Reproducibility for auditors and press

We recommend running panels in that order on a first visit. Activity last gives you a single timeline to screenshot or export mentally against this paper.

SECTION 06

Ticket API reference

The Overview panel documents the ticket endpoint map. Public routes exposed through the sandbox:

OPERATION	METHOD AND PATH
Generate	`POST /api/tickets/generate`
Verify	`POST /api/tickets/verify`
Redeem	`POST /api/tickets/redeem`
Status	`GET /api/tickets/:ticketId/status`
Revoke	`POST /api/tickets/revoke`
Edge architecture	`GET /api/tickets/edge/architecture`
Edge issue	`POST /api/tickets/edge/issue`
Edge verify	`POST /api/tickets/edge/verify`
Edge consume	`POST /api/tickets/edge/consume`
Spent export	`GET /api/tickets/edge/spent/export`
Spent sync	`POST /api/tickets/edge/spent/sync`

QR rendering uses `/api/qr/png` and `/api/qr/svg` with URL parameters. Responses include SVG module counts and EC level for size planning.

SECTION 07

Controls schema and QR boundaries

The sandbox publishes the ticket controls schema in Overview. Key fields:

FIELD	TYPE	MEANING
`maxUses`	integer 0–255	0 = unlimited, 1 = single scan
`entryPoint`	string	Primary allowed gate
`entryPoints`	string[]	Allowed gates; empty = any
`validFrom`	Unix timestamp	Not valid before
`exp`	Unix timestamp	Expiry
`singleEntry`	boolean	Alias for maxUses = 1
`transferable`	boolean	Policy flag; not encoded in QR

Encoded in the QR vs held elsewhere

In the QR: event binding, expiry window, valid-from window, max uses, entry point binding, HMAC signature.

Not in the QR: holder name, email, payment details, seat label (seat may appear in sidecar metadata only).

This boundary is central to privacy-preserving ticket verification and post-quantum audit design. The proof attests to policy compliance on encoded fields. It does not leak holder PII in the scannable payload.

SECTION 08

Minting compact-v3 tickets

We minted with: event `sandbox-event`, tier `standard`, max uses 1, entry point `main-gate`, 24-hour expiry, QR mode `link`. Holder and seat fields were synthetic and did not enter the QR payload.

Response highlights (85ms, Merkle verified):


```
{
  "token": "1DE5B73390AE6A38314C01DAA6000000000000BD0E45",
  "format": "compact-v3",
  "proofDigest": "4e18200fd99be31d0c1e9329d692ee756e30c59137df03b1762f7db22
  "merkle": { "circuit_id": "ticket", "event": "verified", "proof_id": "b73
  "verification": { "serverRequired": false, "offlineCapable": true },
  "merkle_validation": {
    "valid": true,
    "root": "eecf2463f9a69cc162ed92ddeece88432cc81da2ec76de67c7a4d3ecb511b4
    "inclusion_index": 263
  }
}
```


The QR link mode produced a short URL (`/t/fbbfe9e9`) rather than embedding the full token, keeping module count at 29 modules (222px SVG). Edge mode embeds the token directly for offline scanners.

Registry fields on mint: `uses: 0` , `usesRemaining: 1` , `revoked: false` . These update on redeem and consume.

SECTION 09

Verify, redeem, revoke, status

Verify with entry point

Sending token, event `sandbox-event` , and entry point `main-gate` returned `valid: true` in 77ms. Policy block showed `allowed: true` , one use remaining.

Status without entry point

Querying ticket ID `b73390ae` alone returned `allowed: false` , reason `entry_point_denied` , despite the ticket being cryptographically valid. This catches a common integration bug: forgetting to pass the gate context on status polls.

Redeem and revoke

Redeem decrements uses and can trigger Merkle events on consumption. Revoke accepts ticket ID from mint and marks `revoked: true` in registry. We kept our ticket active to test edge consume separately, but all three buttons are available in one panel for lifecycle testing.

> Integration note: tickets bound to an entry point must send that gate on verify and redeem. The sandbox enforces this identically to production CMS behaviour.

SECTION 10

Edge: stateless_at_edge in practice

The edge model is documented in the sandbox schema as `stateless_at_edge`. Summary: the QR code is the proof, not a database lookup key.

PROPERTY	TRADITIONAL QR TICKET	AFFIXIO EDGE
QR role	Database key	Cryptographic proof
Server required	Yes	No
Single point of failure	Yes	No
Screenshot/scalp risk	High	Mitigated by spent proofs
Personal data in code	Often yes	No
Offline scanning	No	Yes

Edge flow (from schema)

1. Scanner reads QR.
2. Crypto and policy verified on device.
3. Spent-proof store checked locally.
4. Green tick or rejection with reason.
5. Consume marks proof digest as spent.

Our edge issue run

Event `sandbox-edge`, gate `gate-a`, max uses 1, 12-hour expiry. Issue: 378ms. Token `CF5C4F51F8F36A3788B001457E0000000000C5718E60`. QR encodes 44 token characters directly (version 2, 25 modules).

Edge verify at gate-a: admitted, 114ms, `offline: true`, `serverRequired: false`, Merkle circuit `edge`, inclusion index 264.

SECTION 11

Spent proofs and gate sync

Anti-scalping depends on spent-proof recording. On consume, the proof digest enters the spent store. A duplicate scan of the same digest fails even if the cryptographic signature is still valid.

The sandbox exposes:

- **Consume:** marks digest spent, decrements uses.
- **List spent:** local spent-proof export for the session gate.
- **Spent export/sync API:** `GET /api/tickets/edge/spent/export` and `POST /api/tickets/edge/spent/sync` for gate-to-gate reconciliation.

For multi-gate venues, sync lets secondary gates import digests consumed elsewhere without centralising admission through one online verifier. That is the operational bridge between offline-first verification and eventual consistency across scanners.

See also WP-021: Double-Spend Prevention for the broader proof-digest model.

SECTION 12

QR generation

The QR panel supports URL mode. We generated a code targeting `https://www.affix-io.com/` in 77ms.


```
{
  "mode": "url",
  "url": "https://www.affix-io.com:3000/q/92a111c2",
  "proofDigest": "8f0172be02fc6952215391c337c08b26e10f547a5378bb4a93660c98e
  "merkle": { "circuit_id": "qr_url", "event": "verified", "proof_id": "92a
  "merkle_validation": { "valid": true, "inclusion_index": 265 }
}
```


Ticket QRs and URL QRs share the audit path. Content provenance and tamper-evident link issuance both produce Merkle leaves. Teams evaluating post-quantum content attestation or synthetic media labelling can test the issuance step here before connecting their own targets.

SECTION 13

Identity verify and ML-DSA-65

The ZK panel separates identity verify from generic circuit prove/verify. Identity verify evaluates rule sets against supplied records and returns a yes/no decision with attestation.

We submitted synthetic records:


```
[{"policy_ref":"POL-7A3C","segment_code":"A1","sequence":"001",
  "group_code":"GRP-ALPHA","status_flag":"active"}]
```


Sector: `verification`. Result in 618ms:

- `decision: yes`, `circuit_id: kyc`, `engine: noir`
- `proof_digest` and `proof_ref` for cross-reference
- `return_value: 0x01`
- Merkle inclusion index 266

Attestation block (post-quantum layer)

```
{
  "signed_at": "2026-06-20T18:46:56.293Z",
  "payload_digest": "550624b74d9bcc552f807864cde25030a35c17b4fea22be74b999c
  "algorithm": "ML-DSA-65",
  "mldsa_signature_b64": "fWL2Khtf7LkSj6g1kOdVCw50hfF1…"
}
```

ML-DSA-65 is the NIST level-3 parameter set from FIPS 204. Signature size is roughly 3.3 KB. Here it signs the payload digest of the verification outcome.

Why this matters for post-quantum migration: hybrid TLS protects the HTTP session. ML-DSA on the attestation protects the decision record itself. An adversary harvesting logs years from now faces lattice-hard forgery on the anchor even if classical signatures on adjacent systems age out. That is the complement to channel-only PQC described across WP-004 and WP-005.

SECTION 14

ZK circuit catalogue

The sandbox lists every public circuit available on the ZK API. Full catalogue from Overview:

`audit_proof` , `composite` , `consent_verification` , `cross_address_proof` , `cross_age_range` , `cross_biometric_match` , `cross_credit_score_range` , `cross_data_consent` , `cross_employment_status` , `cross_income_bracket` , `cross_mfa_verification` , `cross_relationship_verification` , `cross_timestamp_proof` , `edu_academic_eligibility` , `edu_age_admission` , `edu_alumni_status` , `edu_attendance_threshold` , `edu_degree_completion` , `edu_enrollment_verification` , `edu_financial_aid` , `edu_library_access` , `edu_prerequisites_met` , `edu_research_grant` , `eligibility` , plus `kyc` on the identity path and `yesno` as the simplest prove/verify loop.

yesno prove and verify loop

Prove fields: `{"condition_greater1":"1","condition_greater2":"1","condition_greater3":"1"}`.

- **Prove:** 464ms, proof_id assigned, proof hex returned (large blob), Merkle index 267.
- **Verify:** 334ms, `decision: yes`, same `proof_digest`, ML-DSA-65 attestation on verify path.

The proof object is tens of kilobytes. The Merkle leaf is 32 bytes. Storage and audit systems should anchor digests and signatures, not raw proofs or witnesses. That is the practical shape of zero-knowledge compliance at scale.

Circuit complexity drives latency. Ticket crypto stays sub-100ms. Identity kyc with Noir proving sits in the hundreds of milliseconds. Plan gate UX and batch proving accordingly.

SECTION 15

Reading merkle_validation

Every audited response includes a common structure. Learn these fields once, apply everywhere:

FIELD	MEANING
`valid`	Inclusion proof verified against published root
`root`	Merkle root at time of commit
`leaf_hash`	Hash of this operation's audit leaf
`circuit_id`	ticket, edge, qr_url, kyc, yesno, etc.
`event`	Typically `verified` on success
`inclusion_index`	Position in the tree (monotonic over time)

Parallel fields appear at the top level on some responses: `proofDigest`, `merkle.circuit_id`, `merkle.proof_id`. Cross-reference `proofDigest` with edge consume spent lists to trace a ticket from mint through admission.

The Merkle panel and Refresh status button pull the current global root. Compare against per-response roots to understand tree growth during your session.

SECTION 16

Latency benchmarks

All timings from a single sandbox session, reported by the UI on each call (Merkle verification included where stated):

OPERATION	CIRCUIT	INDEX	LATENCY
Health sweep	—	—	204ms
Ticket mint	ticket	263	85ms
Ticket verify	ticket	263	77ms
Ticket status	ticket	263	47ms
Edge issue	edge	264	378ms
Edge verify	edge	264	114ms
QR generate	qr_url	265	77ms
Identity verify	kyc	266	618ms
yesno prove	yesno	267	464ms
yesno verify	yesno	267	334ms

Ticket and QR paths sit at gate-friendly latencies. Edge issue is slower because issuance hits more policy and crypto steps. ZK prove/verify scales with circuit size; kyc identity verify is the heaviest operation we ran.

These numbers will drift with load and region. Treat them as order-of-magnitude confirmation that live post-quantum verification is milliseconds to low seconds, not minutes.

SECTION 17

Reproduction checklist

For security researchers, journalists, and integration engineers validating this paper:

1. Open affix-io.com/sandbox in a fresh browser tab.
2. Record the Merkle root in the header before any operations.
3. Mint ticket: event `sandbox-event`, entry `main-gate`, max uses 1. Confirm `merkle_validation.valid` and note inclusion index.
4. Verify with the same entry point. Confirm `valid: true` and latency under 100ms.
5. Status check ticket ID without entry point. Confirm `entry_point_denied`.
6. Edge issue: `sandbox-edge`, gate `gate-a`. Verify with `offline: true`.
7. Generate QR URL. Confirm `circuit_id: qr_url` in Merkle block.
8. Identity verify with the synthetic JSON array from Section 13. Inspect `attestation.algorithm` equals ML-DSA-65.
9. Run yesno prove then verify. Confirm matching `proof_digest`.
10. Open Activity panel. Confirm request log matches operation count.
11. Refresh status. Confirm Merkle root context updated.
12. Clear session. Confirm state wiped on reload.

If any step returns static Merkle data regardless of operations, the environment is not live. In our runs, indices incremented monotonically from 263 to 267 across eight operations.

Where this sits in the PQC stack

Hybrid TLS (ML-KEM) secures transport. PKI migration (ML-DSA certificates) secures long-lived trust anchors. This sandbox exercises the verification and audit layer: zero-knowledge decisions, ML-DSA-65 signed outcomes, Merkle inclusion proofs, and offline-capable edge tokens. Together they address harvest-now-decrypt-later on identity logs, not just ciphertext on the wire.

> Related: WP-030 Stateless PQ ZK, WP-002 PQ Attestation, WP-011 Merkle Audit, WP-003 Proof Not Log, WP-021 Double-Spend Prevention, WP-029 TLS Hybrid PQC.

FREQUENTLY ASKED

Common Questions

Is the sandbox using mock Merkle data?

No. The root is fetched from `api.affix-io.com`. Each success returns `merkle_validation` with root, leaf hash, circuit ID, and inclusion index against the live tree.

What post-quantum algorithms appear in responses?

ML-DSA-65 (NIST FIPS 204) on identity and ZK verify attestations. Inspect `attestation.algorithm` and `mldsa_signature_b64`. Proof generation uses the Noir engine.

Does the sandbox store personal data?

No server-side PII collection. State is `sessionStorage` only. Reload or Clear session wipes everything.

Can I test offline ticket scanning?

Edge issue sets `serverRequired: false` and `offlineCapable: true`. Verify and consume in the Edge panel; export spent digests for sync testing.

How many ZK circuits are available?

40+ including audit, consent, cross-domain eligibility, education, eligibility, kyc, and yesno. The Overview panel lists the full public catalogue.

Do I need an API key?

No signup for the sandbox. Requests use the public `aio_web_demo` credential. Production integrations use separate keys and SLAs.

What is compact-v3?

AffixIO's HMAC-signed compact ticket token format. Carries event, tier, expiry, uses, and entry binding without embedding holder PII in the scannable payload.

Why did status return entry_point_denied?

Tickets with a bound gate require that entry point on verify, redeem, and status context. Cryptographic validity alone is insufficient if the gate argument is missing or wrong.

How does this relate to post-quantum TLS?

TLS protects bytes in flight. This sandbox tests decision attestation and audit anchoring that survive beyond the TLS session and resist long-term forgery with ML-DSA-65.

Can press independently verify claims?

Yes. Follow the reproduction checklist in Section 17. Every claim about latency, Merkle indices, and ML-DSA-65 presence is observable in JSON responses without AffixIO assistance.
