AffixIO Agentic Payments and the Verification Economy

Stakeholder Whitepaper

AffixIO

2026-04-09

AffixIO Agentic Payments and the Verification Economy

Stakeholder whitepaper for retailers, merchants, issuers, platforms, and adjacent roles

Version: 1.1 Date: 2026-04-09 Scope: Public API capabilities (`https://api.affix-io.com`), merchant SDK (`@affix-io/affixiomerchant`), operational flows, and stakeholder-specific deployment patterns

Document notice: This paper describes externally observable interfaces, published documentation, and integration behavior. It does not disclose proprietary cryptographic construction details beyond what is already published in AffixIO merchant documentation and API descriptions. Binary verification outcomes, policy endpoints, and proof references are discussed at the architectural level suitable for procurement, architecture review, and partner onboarding.

Executive Summary

AffixIO operates a decision engine and trust layer positioned at the intersection of payments, eligibility, and autonomous software agents. The production API advertises itself as providing binary yes or no verification, optional verifiable proofs, and eligibility evaluation across multiple sectors including finance, retail, healthcare, government, and education. The same platform exposes routes oriented toward autonomous systems: agent registration, device registration, delegations, trust policies, payment intents, payments, refunds, and reversals, alongside generic verification against named verification programs (identified by `circuit_id` in API examples) such as `agentic-payment-permission`.

For merchants and terminal operators, the `@affix-io/affixiomerchant` SDK implements an offline-capable acceptance path with local anti-replay controls, optional immediate online validation and synchronization against `https://api.affix-io.com`, and hooks for settlement, fraud, audit, and dashboard integration. For artificial intelligence and workflow agents, the combination of permission endpoints, verification calls, and deterministic transaction identifiers supports idempotent autonomous charging patterns without requiring a human to click “pay” for every micro-step, while still allowing enterprises to enforce terminal validation, dynamic offline policy, and server-side approval where configured.

This whitepaper is organized for different economic roles rather than for a single generic “merchant” persona. A national retail chain, a single-location café, a card issuer building a loyalty-led authorization hook, a marketplace that sits between buyers and sellers, and a government benefits distributor each care about different subsets of the same platform. The following sections explain what is available today as reflected in live documentation and OpenAPI (`/v1/openapi.json`), what the merchant SDK actually does on the wire (health ping, terminal authentication, validate, sync batching), and what remains realistically bounded (for example, placeholder or not-implemented routes, dependency on correct key and terminal hygiene, and the distinction between cryptographic eligibility evidence and scheme-level card network authorization).

Live platform snapshot (illustrative): A `GET` request to `https://api.affix-io.com/health` returns JSON indicating service status, a version string, and dependency checks (for example PostgreSQL and Redis connectivity as returned at document preparation time). Published v1 documentation lists authentication via `X-API-Key` (with note that `Authorization: Bearer` may also be accepted for `/api/*` routes in some deployments) and enumerates endpoints for health, OpenAPI, metrics, verification, proof generation and verification, policy decisions, token generation and validation (including hybrid post-quantum/classical modes), and merchant lifecycle operations. Partners should always treat the live OpenAPI document as the authoritative interface contract for version drift.

We wrote this for mixed rooms: someone from stores, someone from payments IT, someone from legal, and occasionally an issuer product manager who only has thirty minutes. Skim the playbooks in Part IV if you know your hat; read Part II and III if you are the one wiring HTTP. Nothing here replaces your PSP contract or your scheme certification workbook.

Part I: Conceptual Model

1.1 What “agentic payment” means in this architecture

In ordinary commerce software, a payment is often modeled as a single imperative call issued by a person-approved UI event. In agentic commerce, software actors (large language model tool loops, deterministic workflows, robotics controllers, or headless services) initiate or complete payment steps. That shift introduces new failure modes: duplicated charges on retry, unclear attribution of who authorized what, policy drift between agent versions, and insufficient audit material when a regulator or card brand asks “show me the decision path.”

AffixIO’s public materials frame the response along three axes:

1. Binary verification: Reduce complex eligibility questions to allowed or not allowed outcomes, with optional proof references suitable for logging and audit (`POST /v1/verify` with `circuit_id` and `identifier`, returning `eligible` and a `proof` field per published schema).

2. Trust and policy services: Expose explicit decision points such as allow, deny, or review for location, device, merchant rules, spending limits, and user eligibility (`/v1/nior/policy/decide` as described in OpenAPI).

3. Agent permission gating: Answer YES or NO for whether an agent may perform a class of action such as payment, refund, subscription management, eligibility checks, or custom actions

(`/v1/nior/agent/permission`).

The merchant SDK then provides a terminal-local execution environment that is deliberately usable when IP connectivity is absent: it validates inputs, applies local replay and exposure policies, generates compact proof material, queues work, and synchronizes in batches (50 transactions per sync request in the current SDK implementation) when the API is reachable again.

None of this replaces basic hygiene: you still need clear ownership when an agent declines, when a terminal queues for a day, and when finance asks why a proof id on a receipt does not match the PSP statement. The APIs give you hooks and ids; your runbook still has to say who gets paged.

1.2 Separation of concerns: eligibility vs. network authorization

A recurring source of confusion in the market is conflating application-level eligibility with card network authorization. AffixIO's public documentation emphasizes stateless verification and privacy-preserving yes or no outputs for many flows. Card scheme authorization still depends on issuer behavior, risk systems, and acquirer pipes. A realistic deployment uses AffixIO layers to answer questions such as:

- Is this agent allowed to initiate a payment for this merchant program right now?
- Does this terminal satisfy merchant policy for offline acceptance volume?
- Is the customer eligible for a benefit-adjusted price without exposing underlying PII to the merchant POS?

Separately, you still route actual fund movement through the appropriate PSP, acquirer, or issuer rails where required by law and scheme rules.

This whitepaper states that boundary explicitly so that issuers and acquirers can map AffixIO to risk and customer experience workstreams without misunderstanding scope.

1.3 Proof references and auditability

The verification response schema published in OpenAPI includes a `proof` string described as a stable proof identifier or hash for audit. Separately, the platform documents `POST /v1/proofs/generate` and `POST /v1/proofs/verify`. From a stakeholder perspective, the operational value is:

- Merchants can attach proof identifiers to internal transaction journals.
- Platforms can correlate agent decisions across microservices without passing sensitive payloads through every log sink.
- Issuers can design dispute packages that include decision references rather than raw customer dossiers, subject to their own legal and contractual constraints.

This document does not unpack internal proof mathematics; partners should rely on AffixIO security review materials and penetration test summaries for their tier.

Part II: Live API Surface (v1)

The following summary is derived from `https://api.affix-io.com/v1/openapi.json` and the HTML documentation at `https://api.affix-io.com/docs` as accessed for this publication. If a discrepancy exists between this PDF and the live specification, the live specification wins.

Treat this section as a map, not a tutorial. When you wire your first integration, keep two browser tabs open: the HTML docs for prose and examples, and the raw OpenAPI JSON for exact field names. We have watched teams argue for an hour over a field that was renamed in a point release; diff the file when you upgrade.

2.1 Identity, agents, devices, delegations

The v1 OpenAPI describes:

- `POST /v1/agents`: Create an agent with `name`, `type` (`llm`, `workflow`, `robot-controller`, `service`), `ownerOrgId`, and optional `metadata`.
- `GET/PATCH /v1/agents/{agentId}`: Retrieve or update agent records.
- `POST /v1/devices` and `GET /v1/devices/{deviceId}`: Device lifecycle for binding physical or logical endpoints into the trust model.
- Delegations: Create, fetch, revoke, and verify delegations (`/v1/delegations` family), supporting delegated authority patterns common in enterprise procurement and managed service providers.

Retailer relevance: A chain can register one agent per store automation profile or per vendor integration, then narrow blast radius by revoking a single delegation without rotating entire API keys.

Issuer relevance: Issuers experimenting with co-branded bots or assistant-led servicing can treat agents as first-class principals with explicit types, improving internal governance reporting.

2.2 Trust policies and trust checks

OpenAPI lists:

- `POST /v1/trust/check`: Trust decision.
- `POST /v1/trust/policies`: Create trust policy.
- `GET /v1/trust/policies/{policyId}`: Fetch policy.
- `POST /v1/trust/policies/{policyId}/evaluate`: Evaluate policy.

Platform relevance: Marketplaces can encode seller tier, chargeback rate, or category restrictions as trust policies evaluated before an agent is allowed to create a payment intent.

2.3 Payment intents and payments

The payment intent state machine described in OpenAPI includes:

- Create intent, fetch intent, authorise, capture, cancel.
- `POST /v1/payments`: Create payment, with `409` documented for trust check failure.
- `GET /v1/payments/{paymentId}`: Fetch payment.
- `POST /v1/payments/{paymentId}/refund` and `/reverse`: Lifecycle completions with documented conflict responses for wrong state.

Merchant relevance: This mirrors how many merchants already think about auth and capture splits for hotels, rentals, and high-value goods, except the gate can be tied to trust and agent permission outcomes.

Acquirer relevance: Acquirers can map AffixIO payment intent stages to their own gateway tokens and settlement files, using AffixIO as a policy and proof layer upstream of their existing APIs.

2.4 NIOR policy, offline trust scoring, and privacy-preserving proofs

OpenAPI groups several “NIOR” endpoints (names and descriptions as published):

- `POST /v1/nior/policy/decide`: Policy decision point returning `allow`, `deny`, or `review` with reasons and a `decisionId`.
- `POST /v1/nior/offline-trust/score`: Offline trust scoring with `safeToProceed`, numeric `score`, and reasons, explicitly framed for terminal risk, device integrity, pattern anomaly, and offline fraud signals.
- `POST /v1/nior/proofs/query`: Privacy-preserving proofs returning boolean results for predicates such as over-18, eligible benefit, group membership, prior verification, age range, residency, or custom, without returning personal data in the documented response shape.
- Terminal capabilities: List and check terminal capabilities for eligibility checks, offline rules, merchant policy, and secure device verification.
- `POST /v1/nior/agent/permission`: YES or NO permission for agent actions (`payment`, `refund`, `subscription`, `eligibility_check`, `custom`).
- Merchant rules: List, create, and evaluate installable modules (examples named in OpenAPI: student discount, benefits acceptance, location-based, loyalty eligibility).
- `POST /v1/nior/compliance/check`: Offline compliance check returning eligible or not eligible with structured checks, documented as merchant-run for age-restricted sales, sanctions, region, device trust.

Government and education buyers often need binary eligibility without building nationwide identity pipes into every merchant POS. The above endpoints describe a contract-level way to ask constrained questions.

2.5 Generic verification and circuit catalog

- `POST /v1/verify`: Evaluate a verification request against a named `circuit_id` (example in docs: `agentic-payment-permission`) with `identifier` (for example `agent:abc123`) and arbitrary `context` JSON (example includes `amount` and `currency`). Response includes `eligible`, optional `proof`, `circuit_id`, `latency_ms`, `logged`, and `meta`.
- `GET /v1/circuits` and `GET /v1/circuits/{circuitId}`: Discovery and pricing metadata for circuits.

Published marketing copy on the API root mentions 100+ pre-built circuits spanning KYC, age, employment, income, and compliance. Procurement teams should validate the exact catalog available to their organization key, since entitlements may differ by contract.

2.6 Quantum-safe hybrid tokens

Published documentation describes `POST /v1/tokens/generate-quantum-safe` and `POST /v1/tokens/validate-quantum-safe` with explicit algorithm fields (`alg_classical`, `alg_pq`), dual signatures, `verification_policy` values (`classical_only`, `pq_only`, `hybrid_required`), and transport guidance: prefer JSON body or HttpOnly cookies overAuthorization headers for large post-quantum signatures due to proxy header limits.

Issuer and enterprise IT relevance: This is primarily a migration and archival integrity concern for long-lived tokens and attestations, not a day-to-day POS latency optimization. Teams should follow AffixIO’s sizing table (roughly 4–6 KB end-to-end budget for hybrid payloads) when enabling hybrid modes.

2.7 Operational endpoints

- `GET /v1/health` and `GET /v1/metrics`: Operations and observability.
- `POST /v1/api-keys` and revoke routes: key lifecycle.
- `GET /v1/rate-limits`: Rate limit introspection.

2.8 Documented limitations and dual documentation paths

Two documentation layers appear in public materials:

1. The HTML `/docs` page lists both legacy-style paths (`/verify`: `/api-keys`) and v1-prefixed paths, and notes some account-edit endpoints are intentionally omitted from published docs.

2. The v1 OpenAPI file is explicitly titled AffixIO v1 API with server base `https://api.affix-io.com/v1`.

Integrators should standardize on v1 for new builds. The merchant SDK in this repository uses `https://api.affix-io.com` with paths such as `/health`, `/api/merchant/terminals/authenticate`, `/api/merchant/payments/validate`, and `/api/merchant/transactions/sync`, a parallel merchant terminal surface that complements the v1 catalog.

Additionally, `/docs` lists `POST /offline/transactions` as 501 Not Implemented (placeholder). Treat offline batch semantics on that specific route as non-production unless AffixIO announces otherwise.

Part III: Merchant SDK: Wire Protocol and Runtime Behavior

This section summarizes the `@affix-io/affixiomerchant` package as implemented under `/var/www/vhosts/merchant`. It reflects the TypeScript sources `src/merchant.ts`, `src/client.ts`, and `src/sync.ts`.

If you are debugging a stuck queue in production, start with whether `health` is flapping, then terminal auth, then clock skew. Most “mysterious” sync failures we see trace back to one of those three, not to the proof payload.

3.1 Configuration and setup

Developers construct `AffixioMerchant` with:

- `apiKey` (required): minimum length enforced by validation logic; production keys should be loaded from environment or secret managers, never committed to source control.
- `terminalId` / `terminalSecret` (optional but recommended for production): when both are provided, `setup()` attempts terminal authentication against `POST /api/merchant/terminals/authenticate` and stores a terminal-scoped bearer token on the client for subsequent validate and sync calls.
- `baseUrl`: Defaults to `https://api.affix-io.com`.
- `storageDir`: Filesystem location for queue and state when not `memoryOnly`.
- `memoryOnly`: Ephemeral mode suitable for serverless agents and CI; uses in-memory stores instead of disk.
- `offlinePolicy`: Merchant-tunable limits (maximum pending count, value ceilings, time windows, per README and SECURITY guide).
- `hooks`: Extensible integration points documented in `HOOKS.md`.
- `nullifierStorage` / `proofStorage`: Pluggable persistence backends for advanced deployments.

`setup()` also starts a ping monitor that periodically calls `GET /health` (500 ms client timeout in `ping()` implementation) to classify online vs offline status and to trigger background queue flush on transition to

online.

3.2 Payment flow (conceptual ordering)

For each `pay()` invocation:

1. Parse and validate payment fields (card token: positive amount within configured ceilings, ISO 4217 currency, payment method enum, metadata shapes).
2. Invoke `onPaymentInitiated` hook (non-blocking; failures do not cancel payment).
3. Optional blocking `validateTerminal` hook when configured: failure returns a declined result with `error: 'Terminal validation failed.'` or the thrown message.
4. Blocking `getOfflinePolicy` hook invocation: Allows per-merchant dynamic risk tiers; must resolve within hook timeout rules described in hook documentation.
5. Offline guard: Local checks including replay/nullifier semantics, card exposure limits, and cross-terminal velocity style policy without network round-trip.
6. Proof generation: Produces compact proof material and records transaction identifiers, nullifier, and card fingerprint fields needed for server reconciliation.
7. Metadata enrichment: Non-blocking `enrichTransaction` hook merges merchant-specific metadata for reconciliation systems.
8. Online path: If connectivity is considered up, attempts server `validatePayment` when a terminal token exists; on approval, synchronizes immediately via sync API; on success, returns `synced: true`.
9. Offline path: Enqueues transaction locally, persists proof record if configured, returns `queued: true` while still returning `accepted: true` for successful local acceptance.

3.3 Synchronization and batching

`SyncEngine.flush` dequeues all pending transactions and uploads them in batches of fifty per HTTP request to `/api/merchant/transactions/sync`. Partial failures leave failed items for retry; successful items invoke guard confirmation hooks to update local exposure state.

Retail operations insight: Fifty-transaction batches are a practical compromise between mobile radio efficiency and failure blast radius. High-volume festivals may still want to size WAN links assuming periodic batch peaks.

3.4 Security controls (merchant responsibilities)

`SECURITY.md` emphasizes:

- File permissions on `.affixio` state directories (`chmod 700` / `600` patterns).
- Terminal secret distribution through vaults: not source code.
- HTTPS-only production `baseUrl`.
- Safe logging without card tokens, nullifiers, or raw proof payloads in centralized logs.

Card issuer perspective: Issuers remain reliant on merchants not logging tokens from their own hardware abstraction layers. AffixIO documentation aligns with PCI DSS expectations that PAN not appear in SDK logs; merchants must still tokenize at the reader.

Part IV: Stakeholder playbooks

Same product, different jobs. Below is what we tell people when they ask what they actually do with this.

4.1 Single-site retailers and franchised stores

You want the lane to keep moving when the broadband blips, and you want exports that finance can match without a week of spreadsheet archaeology. Run the merchant SDK with a sane offline policy, let it queue, and let it flush when `health` goes green again. Wire `onPaymentAccepted` and `onSyncComplete` into whatever you already use: QuickBooks, Xero, a franchise data lake, or a Postgres table nobody admits to. If you expose tablets in the wild, `validateTerminal` is worth the afternoon it takes to hook to your terminal registry so a random APK cannot present itself as Till 4.

Offline mode does not remove your acquirer. You still settle through your existing rails; you stopped telling customers the computer says no because the router sneezed.

4.2 Multi-site retail chains and venues

Corporate wants visibility; stores want autonomy. Register devices and agents per site so an incident in one city does not force key rotation everywhere. The multi-terminal whitepaper on the site gives real numbers (eight terminals, eighty transactions, percentiles quoted there) as a starting point for capacity talks, not as a guarantee for your WAN. If you run venues, consider tighter offline caps where fraud is worse and looser where shrink is low; `offline-trust/score` can feed that when uplink is flaky.

Measure validate latency and sync latency separately. We have seen healthy internet with a choking validation service behind it; your graphs should show which leg is sick.

4.3 Marketplaces and platform aggregators

You are on the hook for seller quality. Trust policies evaluated before `payment-intents` or `payments` keep half-onboarded sellers out of the money path. The intent lifecycle (authorise, capture, cancel) maps to how you already hold funds between buyer receipt and seller payout. Rules modules cover geographic and loyalty variants without redeploying every seller checkout.

Write plainly in your seller contract who holds PII and who only ever sees a boolean from a proof query. That sentence saves legal a fight later.

4.4 Quick-service restaurants and drive-through

Outdoor antennas drop in rain. Lane readers still need to clear cars. Queue offline, sync when stable. For age-bound add-ons where you cannot photocopy IDs into the POS journal, `nior/compliance/check` is the structured hook; staff training still matters, because no API checks a fake moustache.

4.5 Field sales, pop-up stores, and event merchandising

Tablets in a tent: use persistent storage and watch `queueSize()`. Call `syncNow()` when you pack the truck. `memoryOnly` is for Lambdas and experiments, not for a full day of cashless sales. Encrypt disks, remote-wipe lost devices, rotate terminal credentials when staff churns.

4.6 E-commerce with AI checkout assistants

LLM tool calls retry. Humans double-click. Use transaction ids tied to cart or order identity, not `Date.now()` alone. Call `agent/permission` before `pay()`, and use `verify` with a context that includes the amount and currency you think you are charging. Server-side price authority still lives in your catalogue service; permission is a gate, not a pricing engine.

4.7 Card issuers and program managers

Boolean eligibility and `proofId` style references let you run benefits at POS without mailing dossiers to every merchant. Hybrid tokens matter when your enterprise customers have PQ checklists. Map trust failures on `/v1/payments` to the controls you already run for co-brand apps. Scheme certification is still yours; this document does not replace scheme paperwork.

4.8 Acquirers, PSPs, and payment facilitators

Call `verify` before your gateway if you sell risk as a product. Carry proof ids in metadata fields your processors already propagate. Delegations model sub-merchant agents without sharing one giant API key. Bill API calls on their own line item so pricing stays legible.

4.9 Fraud and risk vendors

`offline-trust/score` and boolean proof outputs are features, not a full model. You still need labels and feedback loops from chargebacks and investigations.

4.10 Audit, compliance, and SOC teams

Persist `decisionId`, `proofId`, and `evaluationId` when your retention policy allows. Fan SDK `onAuditEvent` into SIEM with redaction rules that match your DPA. Metadata without PAN can still be personal data depending on jurisdiction.

4.11 POS hardware and OS vendors

Expose capability checks during provisioning; ship secure storage for terminal secrets. AffixIO cannot fix a device that stores secrets in world-readable files.

4.12 Integrators and agencies

Acceptance: golden tests on `eligible` for fixed fixtures. Load: include fifty-transaction sync batches, not just a single `verify` in isolation.

Part V: End-to-End Reference Flows (Narrative Test Cases)

5.1 Flow A: Online agent checkout with permission gate

1. Marketplace registers agent via `POST /v1/agents`.
2. Before tool execution: orchestrator calls `POST /v1/nior/agent/permission` with `action: "payment"`.
3. If allowed: orchestrator calls `POST /v1/verify` with `circuit_id: "agentic-payment-permission"` and structured `context`.
4. Merchant SDK `pay()` uses tokenized card data from PSP hardware

abstraction.
5. Terminal token path validates and syncs immediately; hooks push results to OMS.

Expected observable artifacts: API permission timestamp, verification `proof` string, merchant `transactionId`, PSP authorization code from external gateway.

If any of those artifacts are missing in your logging pipeline, you will not win an argument with finance or with a card brand. Build the trace before you turn agents on in prod.

5.2 Flow B: Offline festival lane with later reconciliation

1. Lane device loses WAN but continues local `pay()` acceptance.
2. Queue depth monitored; staff instructed to enable backup hotspot if `queueSize()` exceeds merchant threshold.
3. When uplink returns: ping monitor triggers `flush`; batches post to sync endpoint.
4. Finance pulls proof storage export for nightly settlement tie-out.

Expected observable artifacts: Local proof records with `syncedAt` timestamps; sync API per-transaction error array for failures.

Run this drill once in staging with airplane mode on a real device. People believe it when they see the queue drain, not when they read a paragraph.

5.3 Flow C: Hybrid post-quantum enterprise procurement

1. Enterprise IAM mints hybrid token via `tokens/generate-quantum-safe` with `hybrid_required` policy.
2. Token transported in JSON body to internal API gateway.
3. Downstream verification uses `tokens/validate-quantum-safe`.

Expected observable artifacts: token `iat`/`exp`, algorithm fields, validation latency metrics split from business logic latency per AffixIO performance disclosure table in `/docs`.

This path is boring until compliance asks for PQ on internal tokens. Then it becomes the main road.

Part VI: Performance, SLOs, and Benchmark Culture

AffixIO's public documentation cautions engineers to separate native cryptographic timings from ZK proving and proof verification when publishing benchmarks. That discipline matters for agentic payments because tail latency directly affects tool timeouts in LLM orchestrators.

Merchant SDK advertised figures (from README):

- Proof generation target under 333 ms using Web Crypto.
- Nullifier checks under 1 ms in memory.
- Sync batches of 50 transactions.

Independent testing recommendation: For each environment, capture:

- Terminal CPU model and thermal throttling state.
- WAN RTT and packet loss.
- Percentiles for `verify`, `proofs/verify`, and merchant `sync`.

Use those to set orchestrator timeouts so agents fail closed into human review rather than blind retry storms.

Part VII: Error Handling and Idempotency

Published error classes include 400, 401, 404, 409, 422, 429, 501 with meanings enumerated in `/docs`. Merchant-specific guidance:

- 409 on payments indicates trust check failure; orchestrators should not assume money movement occurred.
- Hook timeouts (5 seconds in hook documentation) should be treated as operational defects if they fire frequently. Move slow IO to asynchronous workers.

Idempotency keys: Combine merchant-assigned `transactionId` with stable business keys (order ID hashes) rather than random UUIDs alone when you need cross-session deduplication.

Part VIII: Data Minimization and Privacy Posture

Marketing copy states stateless verification and no PII stored at the platform level. Regardless, merchants still handle tokens and order metadata locally. Privacy officers should review:

- What metadata `enrichTransaction` adds.
- Whether audit logs accidentally include card tokens (explicitly warned against in `HOOKS.md`).

Privacy-preserving proof endpoints are described as returning boolean outcomes only. Legal teams should validate this meets local KYC/AML obligations for each vertical; a boolean alone may be insufficient where document retention is statutory.

Part IX: Regulatory and Scheme Considerations (High Level)

This document is not legal advice. Practical checkpoints:

- PSD2 SCA: Agent-initiated payments may require strong customer authentication depending on jurisdiction and TRA exemptions; AffixIO permission outcomes do not automatically satisfy SCA.
- US Reg E / Nacha: ACH-adjacent flows are outside the scope of the merchant SDK card README; do not conflate.
- Card-present vs card-not-present: Offline acceptance behavior does not change CVM requirements; reader firmware still governs.

Part X: Integration Checklist by Role

Retail IT

☐ Terminal hardware tokenization verified end-to-end.
☐ `.affixio` directory permissions hardened.
☐ Backup connectivity playbook for queue growth.
☐ Proof export integrated into reconciliation jobs.

Issuer product

- [ ] Circuit entitlements confirmed for your tenant.
- [ ] Hybrid token transport sized through gateways.
- [ ] Dispute evidence mapping for `proof` identifiers.

Marketplace engineering

- [ ] Trust policies versioned and audited.
- [ ] Payment intent states mirrored in internal ledger.
- [ ] Seller-facing dashboards show review decisions distinctly from deny.

Part XI: Competitive Positioning and Boundaries

AffixIO combines:

- Broad sector verification catalog (marketing claims 100+ circuits; validate per tenant).
- Agent/device/delegation primitives uncommon in legacy fraud APIs.
- Offline-first merchant SDK with explicit sync semantics.

Boundaries:

- Not a full processor on its own in the sense of scheme licensing: partners bring rails.
- Not a replacement for human supervisors in regulated advice use cases.

Part XII: Field guide (operations, API walkthrough, and vertical notes)

This part is written for people who have to ship the integration, not just read a slide deck. It folds together what used to be a long appendix: same facts, less repetition, more room for judgment calls you will actually make in a war room.

12.1 Stores, groceries, stadiums, and bad weather

If you run shops, you already know the problem: the card reader works, the LAN works, and the internet to the outside world does not. Customers do not care. The merchant SDK is built around that reality. Local acceptance can continue within whatever offline policy you set; the queue grows on disk; when `GET /health` starts succeeding again, the sync engine pushes batches of up to fifty transactions to `/api/merchant/transactions/sync`. Corporate IT should watch queue depth per store the same way they watch disk space, not as a vanity metric. A flat line at zero is not always good (you might be hard-down), but a line that climbs for hours while WAN is supposedly up means something is wrong with credentials, clocks, or your back end.

Grocery is the harsh version of the same story. Margins are tiny, baskets are big, and a thirty-second stall at peak is real money. Tighter offline limits on high-shrink categories (spirits, formula) and looser limits on everyday baskets is a business rule, not something AffixIO invents for you. The `getOfflinePolicy` hook is where you wire that in. Storm outages in rural postcodes are when cumulative offline caps matter; if you hit them, someone in ops made a risk choice, and that choice should be documented.

Stadiums and concerts show up in AffixIO's own multi-terminal whitepaper: eight terminals, eighty transactions, latency percentiles published there. Copy those numbers into your capacity model, then add your own ugliness (one NAT, one egress, everyone syncing at halftime). If you have one bad router, staggering sync or backing off in hooks beats blaming the API.

12.2 Franchises, delegations, and who gets fired after an incident

Franchise systems fight the same battle forever: head office wants consistency, franchisees want local law (alcohol, tobacco, lottery). Delegations in the v1 API are a plain way to say "this integrator may act for this org until we revoke it." Wire revocations to your ITSM ticket when a vendor offboards. When a customer says "the kiosk charged me without consent," your investigator needs a straight line: agent permission log, verify result and proof id if you store it, PSP authorization id, and optional `onAuditEvent` output from the SDK. If any of those is missing, that is not a product gap; that is logging you never turned on.

12.3 Marketplaces, payfacs, issuers, and acquirers

Marketplaces sit between buyer and seller; trust policies and `trust/check` are how you stop a half-onboarded seller from entering the payment path. A `409` on `POST /v1/payments` for trust check failure should feed a seller-health dashboard, not a single Splunk line nobody reads.

Issuers care about benefits and eligibility without shipping government IDs to every corner shop. The `nior/proofs/query` style endpoints return booleans and a `proofId` in the published contract. Your lawyers still decide if a boolean is enough for a given subsidy scheme.

Acquirers and ISOs bundle gateway plus risk. AffixIO fits upstream as a pre-auth gate and as a place to hang proof ids in metadata your processor already carries. Price API calls separately from interchange; mixing them makes finance hate you by month three.

12.4 Government, education, pharmacy counter, BNPL

Public-sector pilots move slowly. Start with read-only dashboards and synthetic traffic before you tie benefits ledgers to new write paths. Pharmacy front-of-store (not the prescription backend) is where age-gated OTC and sharp-object rules show up; `nior/compliance/check` returns eligible or not with reasons. Train staff what "not eligible" means at the register (manager override vs hard stop).

BNPL is credit. AffixIO can gate whether an agent is allowed to start an application; it does not replace underwriting unless you explicitly connect verify circuits to your own models.

12.5 Walking the v1 API in the order teams actually adopt it

Ops wires `GET /v1/health` and `GET /v1/metrics` into monitoring. Apps create orgs and API keys, then register agents with a real `type` field (`llm`, `workflow`, `robot-controller`, `service`) so postmortems are honest about what broke. Devices get their own records so you can suspend one tablet without rotating the world.

Before money moves, most programs hit `trust/check` or evaluate a stored policy. Payment intents follow create, authorise, capture, cancel; if a shopper abandons agent checkout, cancel the intent so you do not leave orphan holds on cards.

The NIOR family is where policy, offline trust scoring, boolean proofs, terminal capabilities, agent permission (`YES/NO`), merchant rules, and compliance checks live. Generic `POST /v1/verify` with a `circuit_id` (docs use `agentic-payment-permission` as an example) is the catch-all when you need a named program without hard-coding every path in your gateway.

Hybrid post-quantum tokens are real in the docs: big payloads, dual signatures, `verification_policy` including `hybrid_required`. Put them in JSON bodies or cookies, not in skinny Authorization headers, or proxies will truncate you.

12.6 What the merchant SDK actually does on the network

Source of truth for this repo is under `/var/www/vhosts/merchant`: `merchant.ts`, `client.ts`, `sync.ts`. The ping uses `GET /health` with a short timeout so flaky LTE tends to queue instead of half-committing. Terminal id plus secret exchange for a bearer token against `/api/merchant/terminals/authenticate`; validate and sync prefer that token when present. Each queued row carries `transactionId`, proof payload, amount, currency, optional customer and method fields, nullifier, and card fingerprint for replay and exposure reconciliation. Hooks time out at five seconds; blocking hooks that call SAP synchronously will ruin your Friday.

12.7 Observability, failure drills, and money conversations

Split dashboards: `/v1/verify` vs `/v1/nior/*` vs merchant sync. Queue depth histograms from hooks beat guessing from support tickets. Chaos drills worth running: revoke a delegation while an agent retries (you want a clean error, not silent success), rotate keys during a maintenance window with a documented rollback, wipe a terminal token and confirm the SDK re-authenticates on boot.

Finance should model three cost buckets: per-call API spend, terminal storage and support, and scheme fees (AffixIO does not make those go away). For QBRs with AffixIO, ask plainly about the `501` placeholder on `POST /offline/transactions` in public docs, circuit catalog changes that break your regression pack, and when hybrid tokens become a default for your sector.

12.8 Questions we get in real calls

Should every microservice call verify? No. Put it on money movement, refunds, and privilege changes. Low-risk reads should stay cheap.

How do we version policy without surprising agents? Track policy versions or hashes where you already use signed envelopes (see the agentic proof gap whitepaper on the site). Feature-flag risky changes.

Does AffixIO replace PSP idempotency? No. Keep idempotency keys on the network side; use deterministic merchant transaction ids for your own ledger.

Can the SDK run in a browser? The README says yes with Web Crypto; do not ship production API keys in front-end bundles.

What queue depth is “bad”? Trends beat absolutes. Alert on sustained growth over hours, not a ten-second spike during flaky handover.

Are demo keys allowed in prod? No. They are rate limited (on the order of a few requests per minute per IP in the published demo response).

12.9 Primary URLs and repo paths

Use these as bookmarks; content drifts, so diff OpenAPI on every release you adopt.

- https://api.affix-io.com
- https://api.affix-io.com/health
- https://api.affix-io.com/docs
- https://api.affix-io.com/v1/openapi.json
- https://www.affix-io.com/whitepapers/ (offline card, multi-terminal, agentic proof gap summaries)
- Merchant SDK: `/var/www/vhosts/merchant/README.md`, `HOOKS.md`, `SECURITY.md`, plus `src/merchant.ts`, `src/client.ts`, `src/sync.ts` for wire behaviour

12.10 Chargebacks, tokens, and customer-facing wording

Scheme chargebacks have their own rules. AffixIO ids help you tell the story (permission at time T, verify outcome, policy decision id), but issuers may still want delivery proof and customer comms. If a shopper denies authorising an agent, your checkout copy and email receipt should have said an automated flow was involved.

Readers must emit tokens, not PAN. The SDK expects a `cardToken` string. PCI scope, P2PE, and key management stay your problem; the SDK docs assume you are not logging secrets.

Plain-language templates you can localise:

- Payment completed with help from an automated assistant. Reference: {transactionId}.
- We need a short manual check; you will hear from us within {SLA}.
- This payment could not complete automatically; try another method or contact support.
- Connection dropped; your payment is saved on this device and will finish when we are back online.

12.11 RACI (example only)

Activity	Store ops	Corp IT	PSP	Issuer	AffixIO
Terminal secret install	A	R	C	I	I
Offline policy tuning	C	A	I	I	C
Agent permission matrix	I	A	I	C	C
Dispute evidence bundle	R	C	R	A	I

R = responsible, A = accountable, C = consulted, I = informed. Rewrite for your actual org chart.

12.12 Risk register (starter)

Risk	Likelihood	Impact	Mitigation
WAN partition at peak	Medium	Lost sales / queues	LTE backup, queue alerting

Agent retry storm	Medium	Customer confusion	Deterministic ids, PSP idempotency
Slow hooks	Low	False declines	Async side queues, cache reads
Sensitive logs	Low	Compliance	Redact tokens and proof bodies
Key leak	Low	Fraud	Vault, rotation, short-lived terminals

12.13 Testing you should actually run

Layer	Unit	Integration	Load
verify	Schema and fixtures	Golden contexts	Burst in staging
merchant pay	Validation	Flip online/offline	Parallel terminals
PQ tokens	Parser	Round trip	Header size limits

12.14 Internal policy snippets worth stealing

1. Any agent that can pay must be registered; prod keys never live in non-prod orgs.
2. Verify contexts stay minimal; PII in context needs DPA sign-off.
3. Queues older than N hours page store lead and IT.
4. Hybrid tokens never go to application logs.
5. Refunds above threshold need a second approval unless permission explicitly allows.

12.15 Vertical go-live checks (short)

Fashion: agent permission on returns; offline limits on promo weekends; hooks into exchanges. Electronics: high-basket review thresholds; disclose agent upsell on receipt. Grocery: LTE per lane; restricted SKU compliance; queue alerts. Hospitality: intent state tests; cancel on booking abort; folio id in metadata. Fuel: keep pump path fast; run compliance only where it cannot block the nozzle. Pharmacy counter: age scripts; no card tokens in SIEM. Marketplace: trust policy versions; 409 visibility; seller comms. Telco: handset subsidy stays in your credit stack; verify only what you contracted. Insurance attach: legal text in UI, not hidden in model state. Vending: device registry; tight offline caps; physical security.

12.16 Benchmarks and marketing numbers

When you see “100+ circuits” or “sub-333 ms proof generation” on the website, treat them as orientation, not a contract for your hardware in a hot back room. Measure your own percentiles before you promise SLAs to a board.

12.17 Scope (again)

This paper stays at the level of published APIs and the merchant SDK. It does not describe internal witness formats, private circuit layouts, or roadmap promises. Your tenant’s entitlements come from the contract and the live OpenAPI diff, not from this PDF.

12.18 Workshop snippets (fictional, for training)

Architect: “We already have a fraud score.” Security: “Scores are opaque; verify gives a named outcome we can cite.” PM: “Agents get one front door before money code runs.”

Store manager: “Receipt says pending sync.” IT: “Funds are not ‘lost’; the device queued a bound record. If it stays pending for days, we debug sync, not the customer.”

SRE: “We hit 429s in a sale.” Account manager path: “Raise quota or cache short-lived permission; add jitter on retries.”

Part XIII: Glossary

- Agent: Registered autonomous actor with typed classification in v1 API.
- Circuit: Named verification program invoked via `circuit_id`.
- Decision engine: AffixIO’s binary eligibility and verification layer described on the public landing page.
- Merchant SDK: `@affix-io/affixiomerchant` offline-capable terminal integration.
- NIOR: Family of policy, compliance, terminal, and agent endpoints in v1 OpenAPI.
- Proof: Opaque identifier or hash referencing verifiable material, per published verify response schema.
- Terminal token: Bearer token obtained after terminal authentication for merchant routes.

Part XIV: Appendix A (curl examples, public patterns)

Health (no auth, dual path examples from public docs):

```
curl -sS https://api.affix-io.com/health
curl -sS https://api.affix-io.com/v1/health
```

Verification (requires key):

```
curl -sS -X POST https://api.affix-io.com/v1/verify \
  -H "Content-Type: application/json" \
  -H "X-API-Key: aff_your_key" \
  -d '{"circuit_id":"agentic-payment-
permission","identifier":"agent:abc123","context":
{"amount":24.99,"currency":"GBP"}}'
```

Demo key issuance (rate-limited):

```
curl -sS -X POST https://api.affix-io.com/api/demo-key
```

Part XV: Appendix B (merchant SDK quick reference)

Install:

```
npm install @affix-io/affixiomerchant
```

Minimal online/offline pay pattern (from published README, adapted):

```
import { AffixioMerchant } from '@affix-io/affixiomerchant';

const merchant = new AffixioMerchant({
  apiKey: process.env.AFFIX_API_KEY!,
```

```
  terminalId: process.env.AFFIX_TERMINAL_ID,
  terminalSecret: process.env.AFFIX_TERMINAL_SECRET,
  storageDir: '/var/affixio/store-001',
});

await merchant.setup();

const result = await merchant.pay({
  cardToken: 'tok_from_reader',
  amount: 42.0,
  currency: 'GBP',
  paymentMethod: 'contactless',
});

console.log({
  accepted: result.accepted,
  transactionId: result.transactionId,
  synced: result.synced,
  queued: result.queued,
  proofBytes: result.proofBytes,
});
```

Agent ephemeral pattern:

```
const agentMerchant = new AffixioMerchant({
  apiKey: process.env.AFFIX_API_KEY!,
  memoryOnly: true,
});
await agentMerchant.setup();
```

Part XVI: Appendix C (hook surface summary)

From `HOOKS.md` (v1.1.2+):

Non-blocking: `onPaymentInitiated`, `onPaymentAccepted`, `onPaymentDeclined`, `onSyncComplete`, `onDoubleSpendDetected`, `onAuditEvent`, `onError`, and non-blocking mode of `enrichTransaction`.

Blocking: `validateApiKey`, `validateTerminal`, `getOfflinePolicy`, and blocking failures in `enrichTransaction` resolution.

Design intent: Merchant business logic integrates without forking the SDK, errors in non-blocking hooks do not roll back an accepted payment.

Part XVII: Appendix D (OpenAPI endpoint index, condensed)

Representative v1 paths: `/health`, `/agents`, `/devices`, `/delegations`, `/trust/check`, `/trust/policies`, `/payment-intents`, `/payments`, `/orgs`, `/api-keys`, `/metrics`, `/rate-limits`, `/nior/...`, `/circuits`, `/verify`, `/proofs/generate`, `/proofs/verify`, `/tokens/generate-quantum-safe`, `/tokens/validate-quantum-safe`.

Always fetch the live JSON for exhaustive path parameters and schemas.

Document History

Version	Date	Notes
1.0	2026-04-09	Initial stakeholder publication
1.1	2026-04-09	Voice edit: removed bold/em dash styling; consolidated appendices; expanded playbooks and field guide

Contact

AffixIO: https://affix-io.com API base: https://api.affix-io.com General inquiries: hello@affix-io.com Security: security@affix-io.com

---

End of whitepaper.