AffixIO Technical Paper
June 2026
affix-io.com

AFFIXIO WHITE PAPER

Cryptographic AI Governance: A Technical Framework

The reference architecture for everything else in this library.

AffixIO | United Kingdom | affix-io.com | June 2026

ABSTRACT

AI governance needs evidence that survives distrust. This framework ties together zero-knowledge proofs, post-quantum signing, and Merkle anchoring into one pipeline for verifiable decisions. Start here if you are new to AffixIO's approach.

CONTENTS

SECTION 1

Introduction

Organisations that deploy AI systems to make or inform decisions face a shared technical problem: they cannot produce cryptographic evidence of what decision was made, under what policy, and at what time. They can produce logs, and logs can be certified and retained, but logs are mutable. Their authenticity depends on the trustworthiness of the logging infrastructure. This dependency becomes significant when decisions are scrutinised in legal proceedings, regulatory investigations, or Freedom of Information requests, because the evidence presented is only as strong as the chain of custody for the log file.

Zero-knowledge proofs change this. A ZK proof is a cryptographic object that demonstrates a computation reached a certain result without revealing the inputs to that computation. Applied to AI governance, a ZK proof can demonstrate that a given piece of content was evaluated by a given circuit, that the circuit produced a specific output (allow or deny), and that this happened at a specific time. The proof does not reveal the content. The proof is bound to the policy version that was active when it was generated. Any party with the public key and the proof can verify the result independently.

AffixIO's production system applies ZK proofs to AI governance. The system runs at a production deployment. Every AI response generated by the platform passes through a policy enforcement layer that hashes the response, evaluates it against a constraint-based circuit, anchors the proof in a Merkle tree, signs the anchor with ML-DSA-65, and appends a verification disclosure to the response before it reaches the user. The total proof pipeline runs before the user sees the response. The audit record consists of audit references, proof digests, and Merkle roots. None of these contain the original response or any user input.

We explain that system with precision. The goal is to be accurate about what the system does and does not do, so that engineers, security architects, and compliance officers can evaluate it on its technical merits.

The paper covers: the mathematical foundations of the zero-knowledge circuits used; the constraint toolkit implementation; the multi-stage attestation pipeline; the Merkle tree construction; the ML-DSA-65 signing; the compliance record service persistence layer; the the client interface layer policy enforcement layer; the source verification and agreement protocol; the safety monitoring layer; the layered information retrieval system; and the regulatory alignment of the system against the EU AI Act, the NIST AI Risk Management Framework, and ISO 42001:2023.

SECTION 2

The Problem with Existing Approaches

The standard approach to AI audit trails is logging. The AI system writes a record of each interaction to a database or log store. The record includes the input, the output, a timestamp, and metadata. When an audit or investigation requires evidence of a decision, the log record is retrieved.

Logging has weaknesses for regulatory and legal purposes that are worth spelling out in detail.

Mutability

A database administrator with sufficient access can alter log records after the fact. Append-only log stores and log integrity services address this to some extent, but they depend on the trustworthiness of the infrastructure operator. There is no cryptographic guarantee available to a third-party verifier that the log record presented today matches the log record as it was written at the time of the original decision. This is the gap that Merkle trees and signed roots close.

Data retention liability

An AI log record that includes the user's prompt and the model's response retains information about the user. Under the UK General Data Protection Regulation and the EU GDPR, retaining personal data beyond what is necessary for the original purpose requires justification. The longer the log retention period required for audit purposes, the larger the data minimisation exposure. If the log database is breached, the breach may expose user communications at scale. A log that contains only cryptographic hashes of content and identifiers, rather than the content and identifiers themselves, changes this risk profile significantly.

No policy version record

If an organisation changes its AI usage policy, log records from before and after the change look structurally identical. There is no way to determine from the log record alone which policy governed the specific decision being

reviewed. This matters when an FOI request or regulatory inquiry asks specifically about the policy that governed a particular decision at a particular time. ZK proofs can embed the circuit version in the proof structure, so that the policy version is part of the cryptographic record, not a separate administrative claim.

Verifiability requires data access

An auditor who wants to verify that a decision was made correctly must have access to the log record, which means access to the raw data. There is no way to provide evidence of a decision to a third party without also providing the underlying data, unless the log is processed and redacted first. ZK proofs change this: the proof and the Merkle root are sufficient for a third-party verifier. No access to the original response is required.

Agent action attribution

Autonomous AI agents that execute payments, export data, modify configurations, or call external services generate an accountability requirement for each action. A log entry records what happened. A ZK proof records that a specific policy was evaluated and produced a specific outcome, with the evaluation bound to the circuit version and anchored in a tamper-evident tree. For regulated contexts such as financial services and government, the distinction between a log entry and a cryptographic proof is material.

The claim here is not that ZK proofs are better than logging in all respects. They are more expensive to generate. They add latency to the response pipeline. They require specialist cryptographic infrastructure (HSMs, circuit libraries, proving systems). The claim is that ZK proofs address specific weaknesses of log-based audit trails that matter in regulated contexts, and that the AffixIO production system demonstrates these properties are achievable at operational scale.

SECTION 3

Zero-Knowledge Proofs: Foundations

A zero-knowledge proof system allows a prover to convince a verifier that a statement is true without revealing why it is true. The formal definition, originating with the work of Goldwasser, Micali, and Rackoff (1985), requires three properties. Completeness: a true statement will cause an honest prover to succeed. Soundness: a false statement will be rejected by an honest verifier except with negligible probability. Zero-knowledge: the verifier learns nothing beyond the fact that the statement is true.

Practical ZK proof systems have developed substantially since the original interactive formulations. The current generation of systems used in production includes zkSNARKs (Succinct Non-interactive ARguments of Knowledge) and zkSTARKs (Scalable Transparent ARguments of Knowledge). AffixIO uses a SNARK-based approach implemented via the constraint language language and the production proving system developed by Aztec.

Arithmetic circuits

In a SNARK, the statement to be proved is expressed as an arithmetic circuit over a finite field. The circuit is a directed acyclic graph of addition and multiplication gates over field elements. A computation that can be expressed as additions and multiplications over a finite field can, with some effort, be expressed as an arithmetic circuit. The circuit takes inputs, performs operations, and produces outputs. The prover, who knows the private inputs (the "witness"), generates a compact proof that the circuit was evaluated correctly on inputs that satisfy the circuit's constraints. The verifier checks the proof against the circuit description (the verification key) without knowing the witness.

The production proving system

proving backend is the proving system used by Noir. It is based on UltraPlonk, a SNARK construction that uses the BN254 elliptic curve. BN254 provides 128-bit classical security. The proving system produces proofs of a few hundred bytes in size, regardless of the complexity of the circuit. Verification of a

BN254-based SNARK takes milliseconds. Proof generation for the circuits used in AffixIO's governance pipeline takes hundreds of milliseconds, which is acceptable for per-response governance where the total pipeline latency budget is measured in seconds.

What the proof proves

For AffixIO's governance circuits, the statement proved takes the form: "I know inputs to the primary policy gate circuit such that primary policy signal evaluates to 1, secondary policy signal evaluates to 1, and tertiary policy signal evaluates to 1." The proof demonstrates this without revealing the values of the three condition inputs. The verifier checks the proof and confirms that the circuit evaluation produced the YES outcome.

The circuit does not prove that the inputs accurately represent the real world. It proves that the evaluation was performed correctly on the inputs supplied. The integrity of the inputs is addressed by the source verification protocol (Section 10) and the ISV identity binding in the policy enforcement layer (Section 9). The circuit proves the computation; the broader pipeline establishes the trustworthiness of the inputs.

Post-quantum status of SNARKs

The BN254-based SNARK construction used in proving backend does not provide post-quantum security. An attacker with a sufficiently powerful quantum computer could, in principle, forge a SNARK proof using Shor's algorithm applied to the discrete logarithm problem on BN254. For AI governance purposes, this is an acceptable trade-off in the current deployment: the zero-knowledge property (hiding inputs) does not require quantum security, and the tamper-evidence property (binding the proof to a Merkle root) is provided by the post-quantum ML-DSA-65 signature on the root, not by the SNARK itself. Section 7 describes the post-quantum signing layer.

SECTION 4

Circuit Design in Noir

constraint language is an open-source domain-specific language for zero-knowledge proofs, developed by Aztec and available under the MIT licence. Its design goal is to allow developers to write ZK circuits without needing to work directly with finite field arithmetic or constraint systems. A policy circuit resembles a typed function: it takes named inputs with types and produces outputs. The constraint language compiler translates the function into the R1CS constraint format, and subsequently into the PlonKish format required by the production proving system.

AffixIO maintains four circuits in production.

The primary policy gate circuit

The general-purpose gate combines multiple binary policy signals into a single pass or fail outcome. Input semantics are configured per deployment. Public documentation describes behaviour categories, not internal witness layouts or constraint implementations.

The identity eligibility circuit

The identity eligibility circuit evaluates KYC eligibility. It takes inputs representing identity match quality, document validity, and risk tier classification, and produces an allow output if all three meet their respective thresholds. The circuit is designed for customer onboarding and account-opening workflows where the organisation needs to record that a KYC check was performed and passed under a specific policy, without retaining the identity data used in the check in the proof record.

The threshold eligibility circuit

The threshold eligibility circuit evaluates conditions combining health status and age against specified parameters. It is designed for use cases where access or eligibility depends on age, health classification, or both: NHS administrative tier checks, age-restricted product or service access, and benefit programme eligibility where health status is a component of the

eligibility decision. The circuit evaluation proves that the requester satisfied the relevant conditions without revealing the specific age value or health status fields.

The eligibility circuit

The eligibility circuit evaluates programme eligibility against a published criteria set. The circuit is parameterised by criteria values rather than having those values hardcoded, which means the same circuit structure can serve different programmes by changing the parameters. A policy update within a programme can be represented by a parameter change, and the version of the parameters can be recorded in the audit reference, allowing the audit trail to distinguish decisions made under the old criteria from decisions made under the new ones without changing the circuit itself.

Circuit implementation and IP

All four circuits are written in constraint language and compiled to the production proving system. The circuit implementations are part of AffixIO's core intellectual property and are maintained as trade secrets. A patent protection is pursued for core attestation methods.

A circuit evaluation proceeds as follows. The calling application or policy enforcement layer constructs the witness: the full set of input values including private inputs and public inputs. The witness is submitted to the attestation service. The policy circuit is evaluated over the witness by the proving system, which generates the compact SNARK proof. The proof and the public inputs are submitted to the verification service. The verification service checks the proof against the verification key, confirms that the public inputs match the proof commitment, and, if valid, proceeds to the Merkle anchoring step.

SECTION 5

The Multi-Stage Attestation Pipeline

The production proof pipeline has multi-stages. Each step is executed within the policy enforcement layer's outlet hook before the AI response is delivered to the user. The pipeline is synchronous up to the penultimate stage; the final stage (upstream sync) runs asynchronously in a separate process.

Stage 1: Content hashing

The AI response text is hashed with SHA-256 to produce a 64-character hexadecimal content hash. Any URLs or source citations present in the response are extracted at this step using a regular expression pattern. The content hash is the primary identifier for the response in the audit trail. Two identical responses will produce the same content hash; this is expected and treated as a match in the record service's deduplication logic.

Stage 2: Record registration

The content hash and any associated URL metadata are submitted to the compliance record service via an authenticated API call. The record service records the hash under the tenant's namespace, associates the extracted URLs with the hash record, and returns a registration confirmation. This step establishes the content hash in the audit trail before the proof is generated, providing a timestamp anchored to the record service's system clock for the moment of response generation. The registration record is the earliest timestamp in the audit trail for any given response.

Stage 3: Source verification

Cited URLs are fetched, normalised, and compared against response claims using exact and approximate matching strategies. Match thresholds and timeout budgets are deployment parameters rather than fixed constants. Results feed the policy circuit as binary signals without retaining page content in the audit record.

Stage 4: Circuit evaluation

The policy enforcement layer constructs the witness for the primary policy gate circuit based on the source verification results from an earlier stage. The three binary condition inputs encode: whether at least one source was verified (primary policy signal), whether the majority of sources were verified (secondary policy signal), and whether no contradiction signal was detected in any source (tertiary policy signal). The witness is submitted to the attestation service. The primary policy gate circuit is evaluated and a compact SNARK proof is generated.

Stage 5: Merkle anchoring and signing

The proof is submitted to the verification service. The endpoint confirms the proof is valid against the verification key, inserts the proof digest as a new leaf in the Merkle tree using SHA-256 sorted-pair hashing, computes the new Merkle root, and submits a signing request to a certified HSM. The HSM performs the ML-DSA-65 signing operation on the leaf insertion record and returns the signature. The verification service returns a structured response containing the attestation_id, proof_digest, tree_root, tree_leaf_hash, and attestation record.

Stage 6: Audit record submission

The proof digest and associated metadata are submitted to the compliance record service via the audit endpoint. Metadata includes the audit reference, Merkle root, source verification results for each URL, deployment identity fingerprint, and runtime environment fingerprint. The record service records the proof in the tenant's audit trail and publishes the Merkle root to the append-only roots log. This is the step at which the proof becomes part of the durable audit record.

Stage 7: Verification disclosure

The policy enforcement layer constructs a collapsible HTML badge containing the audit reference, proof status, source verification results for each cited URL, the Merkle root at time of anchoring, the circuit name and outcome, and a link to the public audit record. The badge is implemented as an HTML `details` element and appended to the AI response. The user can expand the

badge to inspect the proof details. The badge is rendered before the response reaches the user: it is part of the response content, not a post-delivery annotation.

Stage 8: Upstream synchronisation

The anchoring relay runs as a separate process and polls the compliance record service at regular intervals for proof records that have not yet been synchronised to the upstream tree. For each unsynced proof, the daemon submits the proof digest to the verification service at the upstream attestation endpoint. The upstream tree adds the proof digest as a new leaf, recomputes its root, and returns the new upstream Merkle root. The daemon records this upstream root against the proof in the local record store, marking it synced. If the upstream API returns a conflict indicating the proof digest has already been submitted (a double-spend condition), the daemon queries the upstream tree for the existing record and marks the proof synced against the upstream root from that earlier submission.

SECTION 6

Merkle Tree Construction

The Merkle tree used by AffixIO is a binary hash tree where each leaf is the SHA-256 hash of a proof digest, and each internal node is the SHA-256 hash of the concatenation of its two child hashes, sorted in ascending order before concatenation. The sorted-pair scheme ensures that the Merkle root value is not sensitive to the left-right ordering of children: the same set of leaves produces the same root regardless of the order they are assembled, which simplifies inclusion proof verification and allows auditors to reconstruct any subtree without needing to know insertion order.

Full recomputation

The local Merkle tree is rebuilt from the complete leaf set on each proof insertion. This is a full recomputation rather than an incremental update. Full recomputation guarantees that the root accurately reflects the complete leaf set at each point. For the current proof volumes in production, full recomputation completes well within the proof pipeline latency budget. At higher proof volumes, an incremental update scheme would be substituted without changing the tree structure or verification properties.

Inclusion proofs

An inclusion proof for a given leaf consists of the leaf hash and the sequence of sibling hashes that form the Merkle path from the leaf to the root. Any party with the leaf hash, the sibling path, and the root can verify inclusion by iteratively hashing sorted pairs from the leaf up to the root and confirming the result matches the published root. The root is published and signed by the HSM, so the verification chain terminates at the ML-DSA-65 attestation. An auditor does not need to trust AffixIO, access AffixIO's systems, or query any AffixIO endpoint to verify inclusion, provided they hold the signed root and the inclusion path.

Dual anchoring

The system maintains two trees in parallel: a local tree updated on each proof submission, and an upstream tree at the upstream attestation endpoint. The local tree is the authoritative record for the individual deployment. The upstream tree aggregates proofs across instances and provides a verifier-independent reference. An auditor who queries the upstream tree for a given proof does not need to contact the deploying organisation's instance. The upstream tree represents a canonical record that AffixIO operates separately from any single customer deployment.

Append-only roots log

The roots log on the record service is append-only. Roots are never deleted or overwritten. Each new root is timestamped and appended to the log after each proof anchoring. A root published at time T reflects exactly the set of proofs anchored up to time T. Any retroactive insertion or removal of a proof would require recomputing all roots published after the change, producing different root values that would conflict with the signed attestation records for those roots. This makes the tamper-evidence property of the roots log derivable from the signed attestation records without requiring a dedicated append-only log verification process.

SHA-256 sorted-pair scheme: construction

For two sibling hashes A and B at the same tree level, the parent hash is computed as:

```
// If A <= B lexicographically:
parent = SHA-256(A || B)
// If B < A lexicographically:
parent = SHA-256(B || A)
```

This is the same sorted-pair scheme used in Bitcoin's transaction Merkle tree and described in RFC 9162 (Certificate Transparency v2).

SECTION 7

Post-Quantum Attestation with ML-DSA-65

Signing is the mechanism by which the Merkle tree becomes cryptographic evidence. Without signing, the Merkle tree is a data structure that any party could construct and claim represents a given set of proofs. With signing by a key held in a hardware security module under controlled access, the signed root provides evidence that a specific set of proofs existed at a specific time and that the signing entity attested to it. An unsigned Merkle root is an integrity check; a signed Merkle root is an attestation.

ML-DSA-65 and NIST FIPS 204

AffixIO uses ML-DSA-65, the Module Lattice-based Digital Signature Algorithm at Security Level 3, standardised by NIST as FIPS 204 in August 2024. ML-DSA is based on the security of the Module Learning With Errors (M-LWE) problem. The M-LWE problem is believed to be hard for both classical and quantum computers. FIPS 204 is one of four post-quantum cryptographic standards finalised by NIST in 2024, completing a standardisation process that ran from 2016.

ML-DSA-65 provides 128 bits of classical security and corresponds to Security Level 3 in NIST's post-quantum security classification, offering resistance equivalent to AES-192. The signature size is 3,309 bytes and the public key size is 1,952 bytes. These are larger than ECDSA P-256 (64-byte signature, 64-byte public key), but the size increase is appropriate for long-lived attestation records where the signing algorithm must remain secure for the lifetime of the records rather than only for the duration of the communication session.

ML-DSA is deterministic: the same message signed with the same key always produces the same signature. This is relevant for audit purposes: a claimed attestation can be verified against the public key without requiring access to the private key or the signing infrastructure.

Why post-quantum signing now

Three considerations drive the choice to use ML-DSA-65 in production rather than waiting for broader adoption.

First, harvest-now-decrypt-later attacks are a real threat for long-lived records. An adversary who captures signed attestation records today can store them and attempt to forge signatures once a cryptographically relevant quantum computer is available. If the signatures use a classical algorithm such as ECDSA, those future forgeries would be undetectable. If the signatures use ML-DSA-65, forgery requires solving M-LWE, for which no quantum algorithm provides a significant advantage. Governance records that need to remain valid for years or decades should use post-quantum signatures from the point of generation.

Second, procurement requirements are moving now. Banks, defence contractors, and central government agencies in the UK and the United States have begun including PQC readiness questions in vendor security questionnaires. An answer of "we will migrate when necessary" is substantively different from "we deploy ML-DSA-65 in production today." AffixIO's production deployment provides evidence that can be reviewed, not a roadmap commitment.

Third, FIPS 204 is a stable standard. NIST finalised it after eight years of public review and cryptanalysis. Implementing it now does not carry the uncertainty associated with pre-standardisation algorithms.

a certified HSM and FIPS 140-2 Level 3

The signing key is generated and stored inside an a FIPS-validated HSM cluster in a designated region. FIPS 140-2 Level 3 validation includes the requirement that the hardware is tamper-responsive: physical attempts to access the key material trigger the HSM to zeroise (wipe) the keys automatically. The private key never exists outside the HSM hardware boundary in plaintext form. Signing requests are submitted to the HSM over an authenticated, encrypted channel. The HSM performs the ML-DSA-65 signing operation internally and returns only the signature.

a key management service manages the key hierarchy and access control. key management policies specify which authorised service accounts can request signing operations from the HSM. Access to the signing credential is restricted to the attestation service account. The signing timestamp is recorded in the attestation object and is part of the signed payload, meaning any post-hoc alteration of the timestamp would invalidate the signature.

Verification by third parties

The ML-DSA-65 public key is published. Any party who holds the public key and the signed Merkle root can verify that the root was produced by the holder of the corresponding private key at the stated time. Verification does not require access to AffixIO's systems, the HSM, or any AffixIO credential. Verification is available offline using any implementation of the ML-DSA-65 verification algorithm, including the reference implementation provided by NIST as part of the FIPS 204 standardisation package.

SECTION 8

The compliance record service

The compliance record service is the persistence and multi-tenancy layer of the AffixIO system. It is a application service that sits between the ZK proof engine (which handles circuit evaluation and Merkle tree construction) and the customer's audit infrastructure (which needs per-tenant records, compliance reports, and event integrations). The record service persists proof records, manages tenant isolation, generates compliance exports, dispatches webhooks, and maintains the safety guardrail flag log.

Multi-tenant namespace isolation

Each tenant is assigned a namespace. Within that namespace, the record service maintains a proof audit trail, a source verification log, a monthly usage counter, a guardrail flag log, a webhook configuration, an approval ledger, and a roots publication log. These record sets are isolated at the data layer. A tenant credential is scoped to a single namespace: it can read and write records in that namespace but cannot enumerate, read, or write records in any other namespace. The record service validates namespace scope on every request before accessing any record.

Public and administrative endpoint separation

The oracle's endpoints are divided into public (unauthenticated) and administrative (authenticated, internal-only) sets. Public endpoints return information that AffixIO has designated as publicly readable: record service status, health check data, the audit log (most recent records, no PII), and the published-roots log. Administrative endpoints handle tenant provisioning, flag management, webhook configuration, and compliance export. Administrative endpoints are accessible only from within the internal network; the reverse proxy layer blocks administrative path prefixes externally, and the record service applies a secondary check to reject any administrative request that arrives via the proxy path, providing defence in depth.

Compliance export

On demand, the record service generates a complete signed record of all proof activity in a namespace. The JSON export contains, for each proof record: the audit reference, the proof digest, the Merkle root at time of anchoring, the source verification results for each URL (verified or not, exact or approximate), the deployment identity fingerprint, the runtime environment fingerprint, and the timestamp. The HTML compliance report renders this information in a format suitable for regulatory submission, including summary statistics (total proofs, source verification rate, guardrail flag count, Merkle root history) and a full tabular proof record. Both formats are generated on demand and are available to authenticated tenant administrators.

HMAC-SHA256 webhooks

Per-tenant webhook configuration allows real-time event delivery to the tenant's own infrastructure. The record service dispatches signed event payloads for two event types: `attestation.created`, fired on every successful proof anchoring, and `safety.flagged`, fired on every distress or safety pattern detection. Webhook payloads are signed with a per-tenant HMAC-SHA256 secret. The signature is delivered in the X-Signature header. The tenant verifies the signature by computing HMAC-SHA256 of the raw request body using the shared secret and comparing it to the header value. Delivery is attempted asynchronously; if the tenant endpoint returns a non-200 response or does not respond, the record service retries.

Artifact approval ledger

The approval ledger provides a per-ISV record of which content types or deployment contexts have been explicitly approved for proof generation. When the policy enforcement layer's require_release_approval valve is set to true, the filter queries the approval ledger before generating a proof. If no approval record exists for the current ISV and content type combination, the filter does not proceed to circuit evaluation. This implements a human-in-the-loop approval step for new AI deployments before they begin generating

proofs and entering the audit trail. The approval record includes the approving user's identity hash, the timestamp of approval, and the scope of the approval.

Usage tracking

The record service maintains per-tenant proof counts, source verification call counts, and guardrail event counts on a monthly basis. Usage data is available to tenant administrators via the tenant portal and is included in compliance exports. Monthly usage counters support volume-based commercial arrangements and give tenant administrators visibility into their governance activity without requiring them to query the full audit log.

Tenant portal

Each active tenant has a branded portal page accessible at a namespace-specific URL. The portal shows live proof counts (total, today, this week), source verification rate, last proof timestamp, and a direct download button for the compliance report. Tenants can upload their own logo for portal branding. The portal is generated dynamically from the record service's usage data and requires no tenant-side configuration.

SECTION 9

Governance Filter Architecture

The policy enforcement layer is a client middleware hooking the response path. The host application's middleware architecture allows plugins to intercept messages at two points in the message lifecycle: inlet (before the user message is sent to the model) and outlet (after the model generates a response, before it is delivered to the user). AffixIO's primary policy enforcement layer uses the outlet hook, ensuring that every AI response passes through the proof pipeline before the user sees it.

Runtime configuration

The policy enforcement layer exposes administrator-configurable parameters rather than hard-coded behaviour. Typical settings cover attestation service credentials, record service endpoints, tenant scope, transport verification, citation checks, disclosure rendering, timeout budgets, and optional release approval. Exact parameter names and defaults vary by deployment profile and are not part of the public specification.

Audit reference format

Each proof receives an audit reference: a structured locator that binds tenant scope, message direction, content fingerprint, and generation date. Optional fingerprints for deployment identity and runtime environment can be included without revealing underlying configuration values. The exact field order and delimiter scheme are deployment-specific and intentionally omitted from public documentation.

Execution context attestation

When include_runtime_fingerprint is true, the filter computes a hash of the hostname, Python version, and filter version string and embeds it in the audit reference. If the policy enforcement layer is replaced with a different version, deployed on a different host, or modified without updating the version string, the runtime environment fingerprint changes. This means that audit references produced before and after a filter change are distinguishable in

the audit trail. It also means that an attempt to substitute a different filter binary while keeping all other configuration identical is detectable by comparing runtime environment fingerprints across proof records.

ISV identity binding

The deployment identity setting identifies the organisation that deployed the filter. Its hash is embedded in every audit reference. Two different organisations deploying identical filter configurations with different deployment identities produce different audit references and different tenant record namespace records. A proof generated by one organisation cannot be confused with a proof generated by another, even if they use the same tenant namespace and the same ZK circuit.

Release notes

Recent production releases tighten transport verification by default, move verification disclosures to collapsible UI elements, and treat runtime environment fingerprints as on-by-default audit metadata. Version numbers and parameter names are omitted from public documentation.

SECTION 10

Source Verification Protocol

Source verification is the process by which the policy enforcement layer checks that URLs cited in an AI response actually contain the information the response claims to cite. Large language models produce responses that may cite sources which do not support the stated claim, cite pages whose content has changed since the model's training data was collected, or include URLs that do not exist. The source verification protocol addresses these failure modes at response time rather than relying on training data accuracy.

Retrieval

For each URL extracted from the response, the filter fetches the URL directly using an HTTP GET request with a standard browser user agent string. The fetch is subject to the citation timeout setting (default 10 seconds). If the fetch times out or returns a non-200 status, the URL is recorded as unverified. The fetched content is processed by a content extraction component, an open-source web content extraction library, which strips navigation elements, sidebars, footers, and other boilerplate from the HTML and returns the clean article text. The content extraction component's extraction is rule-based rather than ML-based, which makes its output deterministic for a given HTML input.

Exact matching

The extracted clean text is searched for a verbatim occurrence of the cited snippet within a bounded text window. Before comparison, both the snippet and the page text are normalised: consecutive whitespace is collapsed to a single space, leading and trailing whitespace is stripped, and Unicode normalisation (NFC) is applied. If a bounded text window in the page text contains the normalised snippet exactly, the URL is recorded as exactly verified.

Fuzzy keyword matching

If exact matching does not succeed, the filter falls back to keyword overlap scoring. Keywords are extracted from the citation by removing common English stop words and retaining content-bearing terms. The same extraction is applied to the page text (using a sliding window of approximately the expected citation length, sampled at multiple positions). The fraction of citation keywords that appear in the page text keyword set is computed. If the fraction reaches the configured threshold, the URL is recorded as approximately matched. If neither exact nor approximate matching succeeds, the URL is recorded as unverified.

Verification result encoding

The three verification results (per-URL: exactly verified, approximately matched, or unverified) are aggregated into the three binary conditions for the primary policy gate circuit:

- **primary policy signal:** 1 if at least one source was verified (exactly or fuzzy), 0 otherwise
- **secondary policy signal:** 1 if the majority of sources (more than 50%) were verified, 0 otherwise
- **tertiary policy signal:** 1 if no contradiction signal was detected in any source, 0 otherwise (contradiction detection is described in Section 11)

The ZK proof produced in a later stage encodes these three values without revealing the individual source verification scores, the content of the fetched pages, or the specific keyword overlap ratios. The proof certifies the aggregate verification outcome.

Compliance mode

When the strict_policy_mode valve is set to true, the filter enforces a strict source verification policy: if secondary policy signal is 0 (the majority of sources were not verified), the filter blocks the response before delivery and returns an error to the user. This is appropriate for deployments where unverified AI citations carry regulatory risk (medical information, legal advice,

financial guidance). In standard mode, unverified sources are recorded in the audit trail and reflected in the verification disclosure, but the response is still delivered.

SECTION 11

Source Consensus Verifier

The Source Consensus Verifier is a separate AI tool available to the model through the the client interface layer tool interface. Where the policy enforcement layer performs source verification automatically as part of every response pipeline, the Source Consensus Verifier is invoked explicitly when the model wants to perform a more thorough cross-source consistency analysis. The typical use case is a response that cites multiple independent sources for the same factual claim, where the model or the user wants to confirm that the sources agree.

Cross-source consistency scoring

The verifier fetches each source URL independently and extracts clean text using a content extraction component. For each pair of sources, it computes the keyword overlap between their extracted texts. Sources with high pairwise overlap are considered to be drawing on consistent information. Sources with low pairwise overlap may represent divergent information that the model should flag explicitly. The pairwise overlap scores are included in the verifier's output.

Contradiction detection

Contradiction detection looks for a specific class of text patterns within each fetched source. The detector searches for words from a configured contradiction signal list (terms such as "not", "false", "incorrect", "disputed", "debunked", "misinformation", "inaccurate") appearing within 100 characters of any keyword extracted from the cited claim. When a contradiction signal appears in proximity to a claim keyword, the source is flagged as potentially contradicting the claim. This is not a semantic contradiction detector; it is a proximity-based heuristic. It will produce false positives (flagging sources that use negation in contexts unrelated to the claim) and false negatives (missing semantic contradictions expressed without explicit negation). These limitations are described in Section 18.

Verdicts

The verifier produces a verdict from a five-value set:

VERDICT	MEANING
`ALL_VERIFIED`	All fetched sources confirmed the claim and no contradiction signals were detected in any source
`MAJORITY_VERIFIED`	50% or more of fetched sources confirmed the claim and no contradiction signals were detected
`LOW_VERIFICATION`	Fewer than 50% of fetched sources confirmed the claim
`CONFLICTED`	One or more contradiction signals were detected in proximity to claim keywords in at least one source
`INSUFFICIENT_DATA`	No sources could be successfully fetched and parsed

ZK proof of consensus state

A ZK proof of the consensus verdict is generated and anchored using the primary policy gate circuit, encoding the three binary conditions as circuit inputs. The proof certifies the consensus state at the time of the tool invocation: which verdict was produced, whether the majority threshold was met, and whether contradictions were detected. The proof is anchored in the same Merkle tree as policy enforcement layer proofs, giving the consensus check the same audit properties as any other system decision.

SECTION 12

Distress and Safety Guardrail

The distress guardrail is a safety filter that runs at the highest priority in the the client interface layer filter chain, before all other filters and before any model inference is triggered. Its function is to prevent the AI system from engaging with messages that express crisis language, self-harm intent, or highly sensitive distress signals, and to direct users who appear to be in distress to appropriate professional support services. It was designed for AI chat platforms serving the general public, where the probability of encountering users in crisis is non-trivial.

Pattern library

The guardrail uses a library of compiled regular expressions divided into four severity categories. The critical category covers explicit self-harm language, suicidal ideation, and statements of immediate intent. The distress category covers severe emotional distress signals including expressions of hopelessness, inability to cope, and desire to stop existing. The sensitive category covers descriptions of violence, abuse, and traumatic experiences that may indicate a user in an acute state. The upsetting category covers escalation patterns within an ongoing conversation, including statements that the AI system itself is making the user worse.

The guardrail scans the complete conversation history, not only the most recent message. This allows it to detect escalation patterns that develop across multiple turns: a user who begins with neutral language and gradually moves toward crisis expressions would not be detected by a filter that only scans the most recent message.

On detection: critical patterns

When a critical pattern is detected, the guardrail immediately stops processing and returns a message directing the user to appropriate professional support. The message includes the following UK crisis contacts, all verified for 24-hour availability:

- Samaritans: call 116 123 (free, 24 hours)

- Shout crisis text line: text SHOUT to 85258
- NHS 111: for urgent mental health advice
- Emergency services: 999 if in immediate danger
- CALM (for men in crisis): 0800 58 58 58
- Childline (for those under 19): 0800 1111

No AI response is generated. The model inference step is not reached.

Flag and block mechanism

On detection of a critical pattern, or a configured number of distress-category patterns, the guardrail submits a flag record to the compliance record service. The flag record contains: a SHA-256 hash of the client IP address, a SHA-256 hash of the user account identifier, the tenant namespace, the trigger category (critical, distress, sensitive, or upsetting), a list of pattern identifiers that matched, and a SHA-256 hash of the flagged message snippet. No plaintext message content is stored in the flag record. The IP hash and user account hash are recorded in the record service's block list with a default expiry of 30 days. An alert is dispatched to the operations team.

Subsequent messages from the same IP or user account within the block period receive the support message and are blocked without reaching the model, regardless of content.

Administrative unlock

Blocked users can be unlocked before the block period expires by an administrator via the record service's administrative interface. The unlock operation records the administrator's action and the reason in the flag log. This capability exists for cases where the guardrail triggers on an ambiguous message, or where a user who was in distress has since sought appropriate help and wants to resume using the platform.

Priority and filter ordering

The guardrail runs at priority -1000 in the the client interface layer filter chain. the client interface layer executes filters in ascending priority order (lower numbers run first). Priority -1000 ensures the guardrail runs before any other filter, including the main policy enforcement layer and any other plugins installed on the platform. This means the safety check cannot be bypassed by filter ordering and does not depend on the configuration of other filters.

SECTION 13

Layered Information Retrieval

retrieval service is the information retrieval service backing the AI platform at a production deployment. It provides two tiers of retrieval with different latency and quality profiles, designed so that the AI model receives fast initial results for its response and richer content is available for source verification and subsequent queries.

Level A: Fast retrieval via a federated retrieval component

The first tier uses a self-hosted a federated retrieval component instance. a federated retrieval component is an open-source privacy-respecting metasearch engine (AGPL licence) that aggregates results from multiple configured search providers. Queries are submitted to the local a federated retrieval component instance, which fans them out to configured providers and returns aggregated rich snippets (title, URL, and a short text extract). The round-trip time for a tier-1 query is approximately 800 milliseconds under normal conditions.

News detection runs on each incoming query. A set of keyword patterns identifies queries that require current information (terms like "currently", "right now", "today", "latest", "breaking", "live", "trending"). Queries that match the news detection pattern are submitted to a federated retrieval component with a same-day time range filter applied, which causes a federated retrieval component to prioritise results from the past 24 hours across its configured providers.

Tier-1 results are cached in a local cache store with a time-to-live of 300 seconds (5 minutes) for standard queries and 900 seconds (15 minutes) for news queries. Cache hits are served in under 10 milliseconds.

Level B: Full-page enrichment

The second tier performs full-page text extraction and BM25 ranking for the source URLs returned by tier 1. For each URL, the full page is fetched and processed by a content extraction component, which strips navigation, boilerplate, sidebars, and non-article elements, returning clean article text.

The clean text is split into chunks of approximately 300 characters, and the chunks are ranked against the original query using BM25 Okapi, a probabilistic ranking function that scores term frequency against inverse document frequency across the chunk collection. The top-ranked chunks from each source provide the richest basis for AI response generation and source verification.

Up to six source URLs are processed per query, with up to three text chunks extracted per source. Full-page cache entries have a time-to-live of 3,600 seconds (1 hour). News query cache entries use a 900-second (15-minute) TTL. Tier-2 processing runs asynchronously: the API response to the model includes tier-1 snippets immediately, and the cache is updated with tier-2 results in the background. A subsequent query for the same search term returns the richer tier-2 content from cache.

Privacy properties

All search queries are submitted to the a locally hosted retrieval component instance rather than directly to external search providers. The retrieval component's privacy model prevents individual queries from being associated with a user identity at the provider level. The retrieval service service does not log query content to persistent storage beyond the local cache entries. Cache entries are keyed by query hash, not by user identity.

SECTION 14

Security Architecture

The security architecture of the production system addresses several distinct threat models across different layers. The layers interact but are designed to be independently defensible: a compromise at one layer should not automatically compromise the others.

HSM layer: root of trust

The ML-DSA-65 private key is generated inside the CloudHSM hardware and never leaves it. Signing requests are authenticated via KMS and submitted over an encrypted channel. Physical tamper attempts trigger automatic key zeroisation. An attacker who gains logical access to the server cannot extract the signing key; they would need physical access to the HSM hardware, which triggers key destruction. This means that even a complete compromise of the application server does not enable an attacker to forge attestation signatures for past or future proofs.

Oracle administrative controls

Administrative endpoints are restricted to internal network access by two independent controls: the reverse proxy layer blocks administrative URL prefixes before requests reach the oracle, and the record service inspects request metadata and rejects any administrative request that arrives via the proxy path. These controls are independent: disabling one does not disable the other. Administrative credentials are separate from the credentials used for proof generation; a credential used to write proof records cannot be used to read other tenants' records or modify record service configuration.

Tenant isolation

Tenant credentials are namespace-scoped. The record service validates namespace scope on every request before accessing any data. Cross-namespace access is not possible using a namespace-scoped credential. The only credentials that can access records across namespaces are the

administrative credentials, which are not exposed externally. This isolation means that a compromised tenant credential cannot be used to enumerate, read, or modify records belonging to any other tenant.

PII minimisation as a structural property

The oracle's record schema does not include fields for plaintext personal data. The fields used to bind a proof to a user are: a 16-character hex prefix of SHA-256(user_id), a hash of the ISV ID, and a hash of the execution context. The AI response content is not stored; only its SHA-256 hash is stored. This is a structural property of the data model, not a configuration setting. Operators cannot accidentally configure the record service to retain plaintext content because the data model does not have a field for it.

Transport security

TLS certificate verification is enforced by default on all external API calls from the policy enforcement layer (verify_transport defaulting to true since the current production release). The reverse proxy terminates TLS for incoming connections to the record service and the AI platform. All credentials are transmitted over TLS-protected channels. There are no plaintext HTTP paths through which credentials or proof data transit in the production configuration.

Secrets management

All credentials, API keys, and signing key references are injected via environment configuration at process startup. No credentials are hardcoded in application source code. Environment configuration files are maintained with owner-read-only access permissions. A credential rotation process is documented and tested: each service can be restarted with new credentials without downtime.

Webhook authenticity

Webhook payloads are signed with HMAC-SHA256 using a per-tenant shared secret. The secret is generated at webhook configuration time and delivered to the tenant over a TLS-protected channel. The signature covers the full raw

request body. Tenants are expected to verify the signature before processing any webhook payload. A payload with an invalid signature should be rejected, preventing spoofed webhook events from triggering actions in the tenant's systems.

SECTION 15

Regulatory Alignment

Three regulatory frameworks are particularly relevant to the AffixIO system: the EU Artificial Intelligence Act, the NIST AI Risk Management Framework, and ISO 42001:2023. The descriptions below reflect the technical alignment of the system as currently deployed. They do not constitute legal advice, and organisations deploying AffixIO for compliance purposes should seek their own regulatory counsel.

EU Artificial Intelligence Act

The EU AI Act came into force in August 2024, with obligations for high-risk AI systems and general-purpose AI models phased in through 2025 and 2026. High-risk AI systems are defined in Annex III of the Act and include systems used in employment, credit scoring, educational assessment, critical infrastructure management, law enforcement, border control, biometric identification, and administration of essential services.

Article 11 requires high-risk AI system providers to maintain technical documentation throughout the system's lifecycle. The documentation must cover the system's general description, design, development process, training data characteristics, testing procedures, capabilities and limitations, and monitoring and maintenance arrangements. AffixIO's compliance export provides a structured, signed record of AI decisions, source verification results, policy versions, and policy enforcement layer configurations in a format suitable for inclusion in technical documentation packages.

Article 13 requires deployers of high-risk AI systems to provide sufficient information about the AI system's functioning so that operators can interpret its outputs and use them appropriately. The verification disclosure appended to every AffixIO-governed AI response provides transparency directly in the interface: the user can see which sources were verified, what the proof status is, and the audit reference for their specific interaction.

Article 14 requires that high-risk AI systems are designed to allow effective human oversight. AffixIO's artifact approval ledger implements an explicit human approval step before new content types enter the proof pipeline. The

distress guardrail prevents the AI system from autonomously continuing conversations with users who appear to be in crisis. The compliance export provides oversight dashboards for authorised reviewers to monitor AI decision patterns.

Article 15 requires appropriate levels of accuracy, robustness, and cybersecurity. The ML-DSA-65 signing and Merkle anchoring ensure that proof records are tamper-evident and cannot be retrospectively altered. The distress guardrail provides a safety control over AI responses to sensitive content. The dual-layer administrative endpoint controls and HSM-based key custody provide the cybersecurity basis for the attestation function.

NIST AI Risk Management Framework

The NIST AI RMF (v1.0, January 2023; updated 2025) defines four core functions for responsible AI governance: Govern, Map, Measure, and Manage. The Framework is non-prescriptive; it defines functions and categories rather than specific technical controls. The following describes where AffixIO's system contributes evidence for each function.

The Govern function addresses policies, accountability structures, and culture of risk management. The policy version embedded in every proof provides a verifiable record of which policy governed each decision. The per-namespace audit trail enables accountability attribution to specific deployments and ISV identities. The approval ledger documents authorisation decisions for new AI deployment types.

The Map function addresses context establishment, risk identification, and risk classification. The deployment identity fingerprint and runtime environment fingerprint in audit references establish the deployment context of each decision. The namespace structure maps AI deployments to organisational units. The compliance export can be structured by context to support risk categorisation activities.

The Measure function addresses systematic assessment of AI system performance and risk. Source verification rates, proof counts, guardrail flag rates, and Merkle root histories in the compliance export provide quantified

measurements of AI governance performance. The ZK proof provides mathematical certainty about specific decision outcomes rather than probabilistic assessment.

The Manage function addresses processes for prioritising, responding to, and recovering from identified AI risks. The webhook integration enables real-time alerting when governance events occur, including guardrail flags. The administrative unlock capability enables rapid response to erroneous guardrail activations. The compliance export supports incident investigation and regulatory reporting.

ISO 42001:2023

ISO 42001:2023 is the first international standard for AI management systems. It follows the high-level structure common to ISO 27001 and ISO 9001, requiring a management system context (Clause 4), leadership commitment (Clause 5), risk-based planning (Clause 6), operational controls (Clause 8), performance evaluation (Clause 9), and continual improvement (Clause 10).

Clause 8 (Operation) requires documented controls for the AI system lifecycle, including risk assessment, impact assessment, and design controls. AffixIO's proof pipeline provides documented operation-level controls: the audit reference establishes which policy circuit was applied to each AI output; the source verification log documents the information sourcing behaviour of the AI system; the verification disclosure makes AI transparency visible to users.

Clause 9 (Performance Evaluation) requires systematic monitoring, measurement, analysis, and evaluation of the AI management system. The oracle's usage tracking (proof counts, source verification rates, guardrail flag rates by namespace) provides quantified measurements. The compliance export generates a structured report suitable for management review and internal audit. The append-only roots log provides evidence that audit records are complete.

Clause 10 (Improvement) requires nonconformance management and continual improvement. Guardrail flag records document safety-relevant events in the AI system's operation. The compliance export includes flag

history that can feed into nonconformance review processes. Policy version tracking in proof records enables retrospective analysis of decisions made under previous policy versions when improvements are made.

SECTION 16

Open Source Foundation

AffixIO's governance system is built on open-source components at every layer. This is a deliberate architectural choice. When the governance of AI decisions must itself be trustworthy, the components that implement it must be auditable. Proprietary black boxes in the critical verification path undermine the credibility of the evidence those components produce.

COMPONENT	LICENCE	ROLE IN AFFIXIO
Noir	MIT	ZK circuit language for all four governance circuits
proving backend	MIT	Proving and verification system for policy circuits
the client interface layer	MIT	AI chat platform; outlet hook filter architecture
a federated retrieval component	AGPL-3.0	Self-hosted metasearch; tier-1 retrieval
a content extraction component	Apache-2.0	Web content extraction; source verification and tier-2 retrieval
an application framework	MIT	HTTP framework for the compliance record service and retrieval service service

Auditability of cryptographic implementation

The cryptographic implementations in the proof pipeline are auditable because they are open source. A security researcher who wants to verify that the SHA-256 hashing in the Merkle tree is correctly implemented can examine the proving backend source code and the constraint language standard library. A cryptographer who wants to verify that the primary policy gate circuit correctly implements the three-condition AND logic can read the constraint language source. A regulator who wants to understand the source verification matching algorithm can read the policy enforcement layer Python source. None of these audits require access to AffixIO's internal systems or proprietary documentation.

Verification without AffixIO

Because the proof format uses open standards (SHA-256, Merkle trees, ML-DSA-65, JSON), verification of any AffixIO proof does not require AffixIO to be operationally available. Any party with the proof digest, the Merkle sibling path, the signed root, and the public key can verify proof inclusion using standard cryptographic tooling. The public key is published. The roots log is publicly readable. This matters for long-lived audit records: the governance evidence must remain verifiable even if AffixIO ceases to operate as a company, or if a customer's AffixIO deployment is decommissioned after they move to a different provider.

Community security maintenance

Open-source dependencies benefit from community security review. Vulnerabilities in a federated retrieval component, a content extraction component, an application framework, and the client interface layer are tracked in public issue trackers and addressed by their respective maintainers. AffixIO's security posture benefits from this community maintenance without requiring AffixIO's engineering team to maintain the security of those components in isolation. Critical security updates to open-source dependencies are typically available within hours or days of public disclosure, compared to weeks or months for proprietary vendor patches.

Open standards for interoperability

The open cryptographic standards used in the system (SHA-256, ML-DSA-65, HMAC-SHA256, TLS, JSON) are specified in publicly available documents and implemented in widely available libraries. A customer who wants to build their own verification tooling, integrate AffixIO proof records into a separate audit system, or port the verification logic to a different programming environment can do so without requiring any proprietary specification from AffixIO.

SECTION 17

Deployment Considerations

The following considerations apply to organisations evaluating AffixIO for enterprise deployment. They reflect the production system's current architecture and the architectural trade-offs involved in scaling it.

Horizontal scaling

The ZK API and compliance record service are stateless in their request-handling logic: each request is processed independently based on the inputs provided. The stateful component is the Merkle tree, which must be consistent across instances in a distributed deployment. The current architecture uses a single authoritative local tree with a separate upstream sync mechanism (the anchoring relay submitting to the upstream attestation endpoint). This provides geographic redundancy (local tree plus upstream tree) without requiring distributed consensus on tree state. For deployments requiring higher local throughput, a shared tree state (backed by a shared state store) would replace the local local tree store while retaining the upstream sync architecture.

HSM availability

a certified HSM clusters require a minimum of two HSMs in separate availability zones, providing high availability for the signing function within a region. Loss of a single HSM in the cluster does not interrupt proof generation. A complete cluster failure (loss of all HSMs in the cluster) would suspend proof generation because the signing step would fail. Proofs already generated and anchored before the failure are unaffected: their signatures remain valid regardless of HSM availability. The failure recovery procedure involves restoring the HSM cluster from a backup or deploying a new cluster and importing the key material from the HSM backup.

G-Cloud deployment requirements

G-Cloud supplier eligibility requires Cyber Essentials Plus certification for cloud services. The Cyber Essentials Plus scheme requires evidence of: network boundary controls (reverse proxy with administrative endpoint blocking satisfies this), patch management (dependency update process), access control (namespace-scoped credentials), malware protection (OS-level controls), and secure configuration (TLS enforcement, no default credentials). AffixIO targets Cyber Essentials Plus certification by month 4 of the operating plan. G-Cloud listing enables procurement by any UK public sector body without a separate tender process.

UK and US regional separation

The primary deployment uses an a certified HSM cluster in a designated region. A second deployment in us-east-1 (Northern Virginia) is targeted for month 16 of the operating plan, providing a separate ML-DSA-65 key pair for US customer proofs. Regional key separation means UK customer proofs and US customer proofs are signed by different keys, satisfying contractual data residency requirements for customers who cannot have their proof signing operations performed outside their jurisdiction. Cross-border proof verification is possible using the appropriate regional public key.

the client interface layer integration requirements

The policy enforcement layer plugin requires the client interface layer the current production release or later (for the stable outlet hook interface). The filter is installed as a Python plugin via the the client interface layer admin interface. No changes to the the client interface layer codebase are required. The filter's valve configuration is applied per-deployment; different the client interface layer instances can use different namespaces, deployment identities, and circuit configurations while connecting to the same AffixIO ZK API and oracle. The filter source code can be reviewed by a customer's security team before installation.

SECTION 18

Known Limitations

The following limitations are characteristics of the current system. They are described here to support accurate evaluation by engineers, security teams, and compliance officers.

ZK proof proves computation, not input accuracy

A ZK proof proves that the circuit was evaluated correctly on the supplied inputs. It does not prove that the inputs accurately represent the real world. If the policy enforcement layer supplies incorrect inputs to the primary policy gate circuit (for example, if the source verification step incorrectly reports a URL as verified when it does not actually support the cited claim), the ZK proof will certify that the circuit was correctly evaluated on those inaccurate inputs. The proof does not detect errors in input preparation. Mitigations include the source verification pipeline (which reduces input errors for source-based decisions) and the audit trail (which allows post-hoc investigation if an input error is suspected).

Source verification is a heuristic

The configurable keyword threshold for approximate matching is a design parameter that trades recall against precision. A lower threshold would increase false positives (reporting sources as verified when they do not actually support the claim). A higher threshold would increase false negatives (reporting sources as unverified when they do support the claim). The threshold is configurable via the citation timeout setting. The exact match is not a heuristic (a bounded verbatim window either matches or does not), but the approximate match is, and the approximate match is the path taken for most citations where the exact phrasing differs between the response and the source.

SNARK is not post-quantum

The BN254-based SNARK used by proving backend provides 128-bit classical security but not post-quantum security. A quantum attacker could, in principle, forge a SNARK proof using quantum algorithms for the discrete logarithm problem on BN254. The ML-DSA-65 signature on the Merkle tree is post-quantum secure, but the SNARK itself is not. For AI governance use cases, where the ZK property is used to hide input data rather than to commit to a value that must remain hidden against quantum adversaries, this is an acceptable trade-off. A future version of the system could substitute a post-quantum SNARK construction (for example, a hash-based system like zkSTARKs) to close this gap.

Contradiction detection is proximity-based

The contradiction detection used in the Source Consensus Verifier is a proximity-based heuristic. It detects contradiction signal words (negation and dispute terms) within 100 characters of claim keywords. It does not perform semantic analysis. It will produce false positives (flagging CONFLICTED for sources that use negation in contexts unrelated to the claim) and false negatives (missing semantic contradictions expressed without explicit negation terms). Users of the CONFLICTED verdict should understand it as a signal that warrants human review, not a definitive determination that the source contradicts the claim.

Guardrail is not adversarially robust

The distress guardrail uses regular expression pattern matching. A user who is aware of the pattern library could express crisis content in ways that avoid triggering the patterns. The guardrail is designed for genuine crisis detection on general-audience AI platforms, not for adversarial evasion resistance. For platforms where adversarial circumvention of safety controls is a material threat, additional controls (semantic classification, human review workflows) should be considered alongside the pattern-based guardrail.

Single-region signing in current production

The production signing key is held in a single CloudHSM cluster in eu-west-2. A regional AWS outage affecting eu-west-2 would suspend proof generation for the duration of the outage. The US region deployment (targeted for month 16) will provide a separate signing path for US-originated proofs. For UK and EU deployments requiring stronger regional availability guarantees, a secondary UK or EU CloudHSM cluster could be provisioned with a separate key pair for failover.

SECTION 19

Conclusion

AffixIO's production system demonstrates that zero-knowledge proofs can be applied to AI governance at operational scale. The multi-stage attestation pipeline runs synchronously before every AI response reaches the user, generating a cryptographic audit trail that is independently verifiable, does not contain the original response content, embeds the policy version, and is signed with a post-quantum algorithm from a FIPS 140-2 Level 3 hardware security module.

The three technical primitives combined in the system address the three primary weaknesses of log-based AI audit trails:

- The ZK proof addresses data retention liability: the audit record proves that the evaluation occurred without storing the inputs.
- The Merkle tree with a signed, published root addresses mutability: any retroactive alteration of the proof set would produce a different root, conflicting with the signed attestation.
- The ML-DSA-65 post-quantum signature addresses third-party verifiability: any party with the public key can verify the attestation without access to AffixIO's systems, and the signature remains valid against future quantum computing attacks.

The system is built on open-source foundations: policy circuits, the production proving system, the client interface layer, a federated retrieval component, and a content extraction component. The proof format uses open cryptographic standards (SHA-256, ML-DSA-65, HMAC-SHA256). Verification of any proof generated by the system does not require AffixIO to be operational; it requires only the public key, the proof digest, and the published Merkle root.

The system is running in production. The proof counts, Merkle roots, and record service health are publicly readable at a production deployment. The audit trail is accumulating continuously. Any proof generated since the system's deployment can be verified against the published Merkle roots.

The limitations described in Section 18 are real. The SNARK is not post-quantum secure. Source verification is heuristic. The guardrail uses pattern matching rather than semantic classification. These are the right trade-offs for a production AI governance system at the current state of ZK proof technology and information retrieval, but they are trade-offs rather than omissions, and they should be understood as such by any organisation considering deployment.

The broader trajectory is clear. The EU AI Act, the NIST AI RMF, and ISO 42001 all point toward mandatory, auditable AI decision trails that go beyond log retention. Zero-knowledge proofs provide a technical foundation for meeting those requirements without the data-retention liability and verification-access problems that log-based approaches introduce. The AffixIO production system represents one specific implementation of that foundation, built with open-source components, open standards, and a verifiable public record.

Related reading

- WP-003: The Proof-Not-Log Paradigm for AI Audit Trails
- WP-004: Real-Time Zero-Knowledge Governance in the AI Response Pipeline
- WP-010: The Open-Source AI Governance Stack

Frequently asked questions

What is cryptographic AI governance?

Using proofs and signed commitments instead of mutable logs as the primary record of AI policy evaluation.

Which standards does it support?

EU AI Act technical documentation, NIST AI RMF measurement, and ISO 42001 control evidence.

Is the stack open source?

Governance-critical components are open; managed anchoring is optional.