AffixIO Technical Paper · WP-004
June 2026
affix-io.com

AFFIXIO WHITE PAPER · WP-004

Real-Time Zero-Knowledge Governance: Synchronous Proof Generation Inside the AI Response Pipeline

Proofs at reply time, not in next week's audit export.

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

ABSTRACT

Batch compliance is too late for live AI. AffixIO runs an multi-stage governance pipeline that finishes a proving backend proof before the user sees the answer. We share latency budgets, failure modes, and when to async the prover.

CONTENTS

SECTION 1

Introduction

AI governance infrastructure is almost universally asynchronous. An AI system processes requests, writes logs, and the logs are consumed by separate governance processes that operate on different timescales: audit jobs that run nightly, compliance dashboards that refresh hourly, regulatory reports generated quarterly. The user receives the AI response before any governance process has touched it. If the governance process subsequently identifies a problem, the response has already been delivered.

This is not a failure of implementation. It is the natural consequence of treating governance as a separate concern from the AI response pipeline. Governance is expensive: log ingestion, policy evaluation, report generation, and audit trail assembly each consume computation time and introduce latency. Adding them to the synchronous request-response cycle would slow responses by seconds. The standard answer is to move governance out of band.

AffixIO demonstrates that this trade-off is not inevitable. By designing the governance pipeline around a lightweight ZK circuit rather than a general-purpose policy engine, and by using production SNARK backend's proof system which is specifically engineered for speed, AffixIO achieves synchronous governance, where the proof is generated before the response is delivered, within a 2-4 second total latency budget that is acceptable for conversational AI applications.

We explain the engineering in detail: the 8-step pipeline, the latency measurement at each step, the optimisation techniques that bring total proof generation time to 400-600 milliseconds, and the properties that synchronous governance provides that asynchronous governance structurally cannot.

SECTION 2

Why Synchronous Matters

The difference between synchronous and asynchronous governance is not merely operational. It is a difference in what the governance record proves.

An asynchronous governance record proves that the governance evaluation was performed at some time after the response was delivered. It cannot prove that the governance evaluation was performed before the response was delivered. In particular, it cannot rule out the possibility that the response was modified between delivery and governance evaluation, because the governance record was created in a different process from the response delivery. If the response was logged correctly, then the governance record is accurate. If not, the governance record reflects the log, not the actual response.

A synchronous governance record, generated within the request-response cycle before the response is delivered, proves that the governance evaluation was performed on the response that was about to be delivered. There is no window between response delivery and governance evaluation in which a discrepancy could arise. The proof digest is computed from the same AI response object that is delivered. The Merkle leaf is created from the proof digest before the response exits the pipeline. The signed root is anchored before the response header is sent.

For regulated AI applications where governance records may be examined in regulatory proceedings or legal discovery, the difference between these two guarantees is substantive. Synchronous governance records are structurally stronger evidence that a governance evaluation took place for a specific response.

> **Key property:** In AffixIO's synchronous pipeline, a response cannot be delivered without a valid ZK proof having been generated for it. The proof gate is embedded in the HTTP response path: if proof generation fails, the response is withheld. There is no configuration option to bypass the proof gate in production.

SECTION 3

The 8-Step Pipeline

Each AI response in the AffixIO platform passes through eight discrete stages before reaching the user. The stages are sequential; each stage's output is input to the next stage. The total elapsed time from receiving the AI model's response to delivering the governed response to the user is 2-4 seconds in production.

STEP	STAGE	COMPONENT	TYPICAL LATENCY	OUTPUT
1	Response capture	Policy enforcement layer	<1 ms	Raw AI response string
2	Source verification	Source checker	150–400 ms	citation_signal binary flag
3	Policy evaluation	an application framework policy service	10–30 ms	Three binary inputs for circuit
4	Witness preparation	constraint language witness generator	5–15 ms	TOML witness file
5	Proof generation	production SNARK backend	400–600 ms	ZK proof bytes
6	Proof digest	compliance record service	1–2 ms	SHA-256 proof digest hex
7	Merkle anchoring	Merkle tree service	2–5 ms	Leaf hash + Merkle path
8	Attestation signing	ML-DSA-65 HSM signer	20–50 ms	Signed root + tracking ref

The total pipeline adds approximately 600–1100 ms to the response latency in production. Given that the underlying AI model response takes 800–2000 ms depending on prompt complexity, the governed response is delivered to the user within approximately 2–4 seconds of the user submitting their query. This is within the latency tolerance established by user experience research for conversational AI interfaces.

Stage 2: Source verification as a circuit input

Source verification is the most variable step in the pipeline. When the AI response contains citations, the policy enforcement layer extracts source URLs and checks each against a list of approved domains and known-reliable sources. The result is a binary `citation_signal` flag: 1 if all cited sources are

verified, 0 if any are unverified or from a disallowed domain. This binary flag is one of the three inputs to the ZK circuit, connecting source quality to the governance record cryptographically.

Steps 4–5: Witness preparation and proof generation

The policy circuit expects three binary witnesses: `citation_signal`, `topic_signal`, and `scope_signal`. Each is either 0 or 1. The witness generator formats these into a TOML file readable by the proving backend backend. The production SNARK backend prover then generates a proof over the BN254 elliptic curve, producing a proof object of approximately 2KB. The proof generation step is the dominant contributor to pipeline latency.

SECTION 4

constraint language Circuit Architecture

constraint language is a domain-specific language for writing zero-knowledge circuits, designed for safety and developer ergonomics. The AffixIO primary policy gate circuit is the primary governance circuit, used for general AI policy evaluation. It is 19 lines of constraint language code and implements a three-input AND gate with a single output.

```
// AffixIO primary policy gate circuit - policy gate
fn main(
  citation_signal: u1,
  topic_signal: u1,
  scope_signal: u1
) -> pub u1 {
  citation_signal & topic_signal & scope_signal
}
```

The simplicity of the circuit is intentional. Each circuit is compiled to a specific current proving scheme proving key and verification key pair. A more complex circuit would be slower to prove. The primary policy gate circuit's

three-input AND structure is sufficiently expressive to encode the core governance policy while being fast enough to generate proofs within the synchronous pipeline's latency budget.

Circuit version control

Each version of the circuit is compiled separately and assigned a unique identifier: `circuit-ref`, `policy-policy-policy-policy-gate-v2`, etc. The circuit identifier is embedded in the proof digest computation. Proofs generated by different circuit versions are distinguishable in the audit trail and cannot be conflated. When the governance policy changes, requiring a new circuit version, all subsequent proofs are generated under the new circuit identifier. Historical proofs retain their original circuit identifier and remain verifiable under the circuit version that was active when they were generated.

SECTION 5

production SNARK backend

proving backend is the proving system used for policy circuits. It implements the current proving scheme proof system, a variant of the PLONK family of proof systems, over the BN254 (alt-bn128) elliptic curve. proving backend is developed by the Ethereum Foundation and is available under the MIT licence. It is the same proving backend used in the Aztec private computation network.

current proving scheme is selected over the older UltraPlonk backend for three reasons relevant to real-time governance. First, current proving scheme proof generation is approximately 20-30% faster than UltraPlonk for circuits of equivalent complexity. Second, current proving scheme proofs are smaller (approximately 2KB vs 4KB for this circuit), which reduces compliance record service write latency and verification bundle size. Third, current proving scheme has a universal structured reference string, meaning the trusted setup is reusable across circuits, simplifying the operational requirements when new circuits are deployed.

Production proving performance

Latency depends on circuit size, hardware, and concurrency. Published figures are representative ranges, not guarantees.

SECTION 6

Latency Budget Analysis

The 2-4 second user-perceived latency budget is allocated across three components: the AI model response time (800-2000 ms), the governance pipeline time (600-1100 ms), and the network overhead (50-200 ms). These components are partially overlapping: the governance pipeline begins as soon as the AI model response is complete, so its latency adds directly to the model response time rather than being parallel to it.

The governance pipeline's 600-1100 ms budget breaks down as follows. Source verification (150-400 ms) is the most variable component, depending on the number of citations in the response and the availability of the source verification service. Proof generation (400-600 ms) is the most computationally intensive and least variable component, as it is bounded by the algebraic operations in the proving backend backend. All other pipeline steps contribute a combined 30-100 ms.

Comparison to asynchronous approaches

An asynchronous governance pipeline that defers proof generation to a background job would add zero latency to the user-facing response. The trade-off is the structural weakening of the governance record described in Section 2. For AI applications where synchronous governance is not required, asynchronous operation with logging remains appropriate. We think regulated AI applications should use synchronous governance where technically feasible, and that the 600-1100 ms overhead is acceptable for that class of application.

SECTION 7

Optimisation Techniques

Three techniques are applied in production to keep the proof generation latency within the 600-1100 ms target.

Circuit artefact caching

The proving backend prover requires two compiled artefacts for each circuit: the proving key and the verification key. Compiling these artefacts from the policy circuit source takes 2-5 seconds per circuit. In production, the artefacts are compiled once at service startup and cached in memory. All proof generation requests within the service lifetime use the cached artefacts. This eliminates a multi-second startup overhead from the per-request latency.

Witness pre-computation

The three binary witnesses (citation_signal, topic_signal, scope_signal) are computed during the source verification and policy evaluation steps (pipeline steps 2 and 3), which together take 160-430 ms. This computation is performed in parallel with the model response being streamed, rather than waiting for the complete model response before beginning witness computation. This overlap reduces the net pipeline overhead by approximately 100-200 ms in typical cases.

Proof streaming

After the ZK proof is generated (step 5), the proof digest, Merkle anchoring, and attestation signing (steps 6-8) can be initiated before the proof generation step fully completes in some configurations. Pipeline overlap can reduce sequential latency when the prover exposes streaming outputs. Exact savings are deployment-specific.

SECTION 8

Governance Badge

At the completion of the 8-step pipeline, the AI response is augmented with a verification disclosure before delivery to the user. The verification disclosure is a compact JSON object embedded in the response metadata, containing the audit reference (a human-readable identifier derived from the proof digest), the circuit identifier (e.g., circuit-ref), the proof outcome (YES or NO), and the Merkle root at the time of anchoring.

In the AffixIO the client interface layer interface, the verification disclosure is displayed as a small indicator below each AI response, showing whether the response was governed (green) and providing a link to the self-contained verification bundle for that specific response. The verification bundle includes all the information needed for third-party verification without querying AffixIO's infrastructure.

> **Verification disclosure contents:** tracking_ref (e.g., ZK-A3F2E1), circuit (circuit-ref), outcome (YES), tree_root (hex), ts (ISO 8601 timestamp). The tracking ref is the stable identifier for the governance event and can be cited in regulatory submissions.

SECTION 9

Failure Modes and Fallbacks

The synchronous proof pipeline can fail at each of its multi-stages. The governance policy determines what happens when a step fails: whether the response is withheld, whether a NO governance decision is issued, or whether the failure is escalated for human review.

Proof generation failure (step 5) is the most critical failure mode. If proving backend returns an error or times out, the pipeline cannot produce a valid proof. In production, a proof generation timeout of 2 seconds is applied. If the timeout is exceeded, the policy enforcement layer issues a NO decision with an error code and withholds the response. The user receives a "governance

check failed" message rather than an unverified AI response. This is a conservative approach: the response is never delivered without a governance record, even if the governance record indicates failure rather than success.

HSM signing failure (step 8) does not withhold the response. If the HSM is unavailable, the proof and Merkle anchoring are complete; only the attestation signature on the root is missing. The response is delivered with the proof and Merkle anchor but marked as "attestation pending" in the verification disclosure. The attestation is completed when the HSM becomes available, and the signed root is appended to the roots log at that point. This is a deliberate trade-off: HSM availability outages should not block user-facing responses, because the proof and Merkle anchoring already provide tamper evidence for the governance record.

SECTION 10

Scaling Characteristics

The synchronous proof pipeline scales horizontally. Each policy enforcement layer instance is stateless: it holds the circuit artefacts in memory but does not share state with other instances. Proof generation is CPU-bound and does not require network calls. The Merkle tree service and compliance record service are shared across instances but are write-optimised: each governance event is a single leaf insertion and a single root publication, both of which are O(log n) operations.

At 100 concurrent governance requests, a four-instance deployment handles the load with 95th percentile total pipeline latency of approximately 1.2 seconds. At 500 concurrent requests, horizontal scaling to 16 instances is required to maintain sub-2-second pipeline latency. The proving backend proof generation step is the binding constraint: each concurrent proof requires approximately one CPU core for 400-600 ms. The policy enforcement layer deployment should be provisioned at one core per expected concurrent proof, with 20% headroom.

SECTION 11

Deployment Requirements

A production deployment of the real-time ZK governance pipeline requires the following infrastructure components.

COMPONENT	MINIMUM SPEC	NOTES
Policy enforcement layer	2 vCPU, 2 GB RAM per instance	CPU-bound on proof generation; scale horizontally for concurrency
proving backend backend	Included in policy enforcement layer	Statically linked; no separate service required
compliance record service	1 vCPU, 512 MB RAM	I/O-bound; handles multi-tenant proof persistence
Merkle tree service	1 vCPU, 1 GB RAM	Maintains append-only tree; state must be durable
HSM (signing)	a certified HSM or equivalent FIPS 140-2 L3	Required for post-quantum ML-DSA-65 signing
Source checker	1 vCPU, 512 MB RAM	Network-bound; rate-limited by upstream sources

The minimum viable deployment for a single-tenant AI governance application is a single compute instance handling all components except the HSM. The HSM is a cloud service and does not require dedicated hardware in the application infrastructure. Total compute cost for a minimum deployment is approximately the cost of one small cloud instance plus the HSM cluster subscription.

SECTION 12

Conclusion

Real-time ZK governance is engineering-intensive but operationally practical. The 8-step pipeline adds 600-1100 ms to the AI response latency, generating a cryptographic ZK proof, Merkle anchor, and post-quantum attestation

signature for every response before it is delivered to the user. This provides governance records with structural properties that asynchronous governance cannot match: the proof is generated on the same response object that is delivered, within the same request-response cycle, and the pipeline cannot be bypassed without blocking the response delivery.

The key enabling technology is the combination of a simple three-input policy circuit and the production SNARK backend prover, which together generate verifiable ZK proofs in 400-600 ms, fast enough to be embedded in a synchronous conversational AI pipeline without unacceptable latency impact. AffixIO operates this pipeline in production at a production deployment, where it has generated governance records for every AI response served since deployment.

Related reading

- WP-003: The Proof-Not-Log Paradigm for AI Audit Trails
- WP-005: Source Verification as a Zero-Knowledge Circuit Input
- WP-001: Cryptographic AI Governance: A Technical Framework

Frequently asked questions

How long does proof generation take?

Hundreds of milliseconds for AffixIO's governance circuits on current hardware, within typical LLM response budgets.

What happens if proving fails?

The response is withheld and a failure proof is recorded, similar to a policy deny.

Can proving run asynchronously?

Yes for non-interactive workloads; regulated chat interfaces usually require synchronous proofs.
