AffixIO Technical Paper · WP-012
June 2026
affix-io.com

AFFIXIO WHITE PAPER · WP-012

The Open ZK Circuit Library: gate, kyc, threshold eligibility, and eligibility Circuits for AI and Identity Governance

The circuits we run in production, documented so you can extend them.

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

ABSTRACT

Policy belongs in code you can audit. This specification covers AffixIO's open constraint language library: gate, kyc, threshold eligibility, and eligibility circuits with witness layouts, proving times, and version rules we use at a production deployment.

CONTENTS

1 Introduction
2 Circuit Design Principles

3 The gate Circuit
4 The kyc Circuit
5 The threshold eligibility Circuit
6 The eligibility Circuit
7 Circuit Version Management
8 Proving Key and Verification Key
9 Proof Generation Reference
10 Verification Reference
11 Extension Patterns
12 Conclusion

SECTION 1

Introduction

A ZK circuit library is to zero-knowledge governance what a standard library is to software development: a collection of well-tested, well-documented, reusable components that eliminate the need to build common functionality from scratch. Building a ZK circuit correctly requires expertise in circuit soundness, constraint system construction, and ZK-specific bug classes (under-constrained outputs, missing range checks, unexpected field arithmetic). Providing a vetted library of circuits for common governance domains lowers the barrier to adoption and reduces the risk of circuit-level vulnerabilities.

AffixIO's four-circuit library covers the most common governance domains for regulated AI and identity applications. The primary policy gate circuit is the general-purpose AI policy gate. The identity eligibility circuit addresses identity eligibility. The threshold eligibility circuit addresses age and health threshold conditions. The eligibility circuit addresses multi-condition entitlement decisions. Together they cover the governance requirements described across WP-003 through WP-009 in the AffixIO whitepaper series.

All four circuits share the same architectural principles: minimal constraints, binary inputs from evaluated conditions, single binary output, clean circuit versioning, and compatibility with the production SNARK backend. The shared architecture means that the Merkle anchoring, attestation signing, and

verification infrastructure is identical for all four circuits: a proof from any circuit in the library can be anchored in the same tree and verified with the same verification bundle procedure.

SECTION 2

Circuit Design Principles

Four principles govern the design of all circuits in the library.

Minimum constraint count. ZK proof generation time scales with the number of constraints in the circuit. All circuits in the library are designed to minimise constraint count while correctly implementing the stated policy. Circuits are sized for synchronous attestation budgets without publishing internal constraint layouts.

Binary inputs only. All circuit inputs are single-bit values (u1 in Noir). This means the circuits evaluate conditions that have already been computed externally: they do not perform arbitrary computation over raw data. The external computation (source verification, identity verification, age calculation) is performed by the policy enforcement layer or adapter layer. The circuit gates the results of that computation. This separation keeps circuits simple and fast, and confines complex computation to off-circuit code that is easier to test and audit.

Single binary output. All circuits produce a single public binary output: 1 (YES/pass) or 0 (NO/fail). Multi-output circuits would complicate the governance record format and the verification procedure. The single binary output is sufficient for all current governance use cases: the question is always "does this event satisfy the policy?" with a yes or no answer.

Explicit versioning. Each circuit version has a distinct identifier (e.g., `circuit-ref`, `policy-policy-policy-policy-gate-v2`). A proof generated under one version cannot be verified under a different version's verification key. The version identifier is part of the governance record, binding each proof to the specific policy version under which it was generated.

SECTION 3

The gate Circuit

The primary policy gate circuit is the primary AI policy gate. It receives three binary witnesses (citation_signal, topic_signal, scope_signal) and produces their logical AND as the output.

Circuit implementation omitted from public documentation.

WITNESS	TYPE	SOURCE	POLICY MEANING
citation_signal	u1	Source verification service	All cited sources are on the approved registry
topic_signal	u1	Topic classifier	The response topic is in the permitted set
scope_signal	u1	Scope checker	The response is within the configured service scope

Version identifiers, constraint counts, and latency figures are deployment-specific and omitted from public documentation.

SECTION 4

The kyc Circuit

The identity eligibility circuit evaluates four identity eligibility conditions. It is used in KYC workflows as described in WP-006.

Circuit implementation omitted from public documentation.

Version identifiers, constraint counts, and latency figures are deployment-specific and omitted from public documentation.

> **Configurable threshold:** The over_age_threshold witness is computed by the adapter layer using a configurable threshold (18, 21, or 25). The circuit does not receive the threshold value; only the binary result of the

> comparison. Changing the threshold requires no circuit change, only an adapter configuration change.

SECTION 5

The threshold eligibility Circuit

The threshold eligibility circuit performs an arithmetic age comparison. Unlike the gate and identity eligibility circuits, it accepts non-binary inputs (birth year and current year as u16) and performs subtraction and comparison inside the circuit.

Circuit implementation omitted from public documentation.

Version identifiers, constraint counts, and latency figures are deployment-specific and omitted from public documentation.

The threshold eligibility circuit is used in the privacy-preserving age verification context (WP-009) and for health condition age eligibility checks (for example, verifying that a user is above the age required for a health screening programme without revealing their exact age).

SECTION 6

The eligibility Circuit

The eligibility circuit is a configurable multi-condition gate. It accepts up to eight binary witnesses and evaluates a configurable subset of them (specified by a bitmask) using AND logic.

Circuit implementation omitted from public documentation.

Version identifiers, constraint counts, and latency figures are deployment-specific and omitted from public documentation.

The eligibility circuit is intended for complex entitlement decisions in welfare, benefits, professional licensing, and similar contexts where multiple independent conditions must be met and the set of required conditions varies by service type.

SECTION 7

Circuit Version Management

Each circuit version is assigned an identifier in the format `{circuit_name}-v{version_number}`. Version numbers are incremented for any change to the circuit source code, including changes to the number or type of inputs, changes to the logic, and changes to the output type. Version numbers are not incremented for changes to documentation, comments, or formatting.

When a new circuit version is deployed, the previous version remains active for proof generation until all existing governance records under the previous version have reached their configured retention period. This means that during a version transition, the policy enforcement layer may need to support proof generation under multiple circuit versions simultaneously. The circuit artefact cache holds compiled artefacts for all active versions.

Historical governance records always reference the circuit version under which they were generated. A regulator examining a historical record can obtain the source code for that specific circuit version from the version-tagged source repository, compile it independently, and verify that the stated circuit identifier corresponds to the stated circuit logic. This provides a complete chain of evidence from the governance record to the specific policy that governed the AI response in question.

SECTION 8

Proving Key and Verification Key

Each circuit version requires two compiled artefacts: the proving key (used to generate proofs) and the verification key (used to verify proofs). Both are derived from the circuit source code via the proving backend compilation process. The proving key is approximately 50 MB for circuit-ref; the verification key is approximately 2 KB.

The verification key is the critical public artefact. Any party who holds the verification key for a specific circuit version can verify proofs generated under that version, without access to the proving key or any other secret material. AffixIO publishes verification keys for all active circuit versions as static files. Verification keys for retired circuit versions are archived and remain publicly accessible for the purpose of verifying historical governance records.

Proving keys are private in the operational sense (they are large and cached on the governance server) but not secret in the cryptographic sense: knowing the proving key does not enable an adversary to generate false proofs. A false proof for a given circuit and verification key is computationally infeasible under the discrete logarithm assumption on BN254.

SECTION 9

Proof Generation Reference

Generating a proof using the AffixIO circuit library requires three steps: witness preparation, proof generation, and proof digest computation.

```
# Witness preparation, proving, and digest derivation follow deplo
# Public docs omit field names, CLI flags, and internal artefact pa
```

The proof digest format is `a composite digest over circuit identity, outcome, and proof material`. The outcome is "1" for YES and "0" for NO. The proof digest is the stable identifier for the governance event and is used as the leaf data for Merkle anchoring.

SECTION 10

Verification Reference

Verifying a proof requires the proof bytes, the verification key for the circuit version, and the public output (the single bit). The verification checks that the proof is a valid current proving scheme proof for the given verification key with the given public output.

```
# Verification uses the published verification key and proof artefa
# Exact CLI invocations vary by prover deployment.
```

Third-party verification of a complete governance record (including Merkle inclusion) requires additionally verifying the leaf hash, the Merkle sibling path, and the ML-DSA-65 signature on the root, as described in WP-011 Section 6. The complete verification bundle for a governance record consists of: the proof bytes, the circuit version identifier, the verification key, the public output, the leaf hash, the Merkle sibling path, and the signed root attestation.

SECTION 11

Extension Patterns

Organisations with governance requirements not covered by the four core circuits can extend the library using two patterns.

Adapter pattern. If an organisation's policy can be expressed as conditions that produce binary outputs, those conditions can be mapped to the existing circuit inputs using a custom adapter. For example, a credit risk assessment that produces a pass/fail binary result can be mapped to the `scope_signal` input of the primary policy gate circuit without any circuit change. The circuit remains the same; the adapter changes. This is the preferred pattern for conditions that can be pre-evaluated externally.

New circuit pattern. If an organisation's policy requires computation that cannot be pre-evaluated externally (for example, a threshold computation over a value that must remain private within the circuit), a new circuit can be

written in Noir. New circuits must undergo a security review by a cryptographer familiar with circuit soundness properties before being deployed in production. The review should check for under-constrained outputs (outputs that do not depend on all circuit inputs as required by the policy) and for unexpected field arithmetic behaviour (field overflow, signed integer handling).

SECTION 12

Conclusion

The AffixIO open ZK circuit library provides four production-ready circuits for the most common AI and identity governance domains. All circuits are written in Noir, proved by proving backend, MIT licensed, and designed for compatibility with AffixIO's Merkle anchoring and post-quantum attestation infrastructure. Proof generation times range from 420 ms (circuit-ref) to 700 ms (eligibility-v1), all within the synchronous AI response pipeline's latency budget.

The library's design principles, minimum constraints, binary inputs, single binary output, and explicit versioning, ensure that the circuits remain fast, auditable, and compatible with evolving governance requirements. New circuit versions can be deployed without invalidating historical governance records. The open-source circuit source code is the machine-readable specification of AffixIO's governance policies, making policy auditing as straightforward as reading a nineteen-line constraint language function.

Related reading

- WP-001: Cryptographic AI Governance: A Technical Framework
- WP-004: Real-Time Zero-Knowledge Governance in the AI Response Pipeline
- WP-010: The Open-Source AI Governance Stack

Frequently asked questions

Which circuits ship today?

gate for policy gates, kyc for identity eligibility, threshold eligibility for threshold checks, and eligibility for composite rules.

Can we fork and extend circuits?

Yes. Circuits are open source with semantic versioning; breaking witness changes bump major versions.

What prover backs these circuits?

production SNARK backend in production, with verification keys published for third-party checking.