UKIYO NETWORK · TECHNICAL WHITE PAPER

Agency for Smart Contracts

Policy-bounded agent execution on Arbitrum

Document type: Technical white paper · Version 3.0

Execution environment: Arbitrum One · Governance anchor: Ethereum L1

Classification: Public

Contents

1. Executive Summary
2. Threat Model: Unbounded Admin and Manual Delay
3. Policy Engine Contract Architecture
4. Allowlist Semantics and Selector Filtering
5. Simulation Gate on Arbitrum Forks
6. Human Escalation and Multi-Sig Design
7. Timeboost and Latency-Sensitive Escalation
8. Key Separation and Off-Model Credentials
9. Agent Runtime and Calldata Construction
10. Sequencer Failure and L2 Resilience
11. Security Analysis and Failure Modes
12. Testing Strategy
13. Governance and Policy Upgrades

1. Executive Summary

The Agency for Smart Contracts is a framework for policy-bounded DeFi agents that sit between live oracle feeds and on-chain markets. Agents watch scores, simulate contract calls, and execute only allowlisted risk actions—pause, reprice, notify, advance. High-impact moves escalate to human multi-sig. The design principle is explicit: agents propose; contracts enforce; keys and limits stay off-model.

The Agency deploys on Arbitrum One alongside SPOOL, with the policy engine and multi-sig co-located for minimal cross-contract latency. Arbitrum's ~250ms block time makes risk-reducing actions fast enough to matter, while Timeboost provides an optional express lane for latency-sensitive pauses and reprices. This white paper specifies the policy engine, allowlist semantics, simulation gate, escalation design, Timeboost usage, sequencer resilience, and testing strategy.

2. Threat Model: Unbounded Admin and Manual Delay

Threat	Description	Agency control
Unbounded admin key	A single key can call any function on any target	Allowlist of targets and selectors
Agent model compromise	Model proposes malicious calldata	On-chain selector filtering; simulation gate
Manual response latency	Humans cannot react at machine speed	Autonomous execution within bounded scope
High-impact autonomous error	Agent executes a large, irreversible action	Value-threshold escalation to multi-sig
Credential leakage	Model or runtime exposes signing keys	Keys held in HSM/MPC; never in model context

3. Policy Engine Contract Architecture

The policy engine is an on-chain contract that acts as a gatekeeper: it allows only approved modules, targets, and selectors to execute transactions. It sits between the agent and the target protocol on Arbitrum One.

```
interface IPolicyEngine {
    struct Policy {
        address target;        // contract the agent may call
        bytes4  selector;      // function selector allowed on target
        uint256 maxValue;      // max native/stablecoin value per call
        uint64  validAfter;    // time window start
        uint64  validUntil;    // time window end
        bool    enabled;
    }

    function setPolicy(bytes32 policyId, Policy calldata p) external onlyGovernance;
    function evaluate(
        address agent,
        address target,
        bytes4 selector,
        uint256 value,
        bytes calldata data
    ) external view returns (bool allowed, string memory reason);
}
```

The engine is stateless with respect to agent strategy: it stores policies, not intentions. The agent supplies the intended call; the engine decides whether that call is permitted.

An empty allowlist authorizes nobody; an empty blocklist authorizes everybody. The semantics are explicit in the contract to prevent misconfiguration.

4. Allowlist Semantics and Selector Filtering

The allowlist is the core security boundary. It is defined at the level of (target, selector), not merely target, because a contract may expose both safe and dangerous functions.

```
mapping(address => mapping(bytes4 => bool)) public allowedSelector;

function evaluate(...) external view returns (bool, string memory) {
    if (!allowedSelector[target][selector]) {
        return (false, "SELECTOR_NOT_ALLOWED");
    }
```

```
    if (value > maxValue[target][selector]) {
        return (false, "VALUE_EXCEEDS_LIMIT");
    }
    return (true, "");
}
```

4.1 Argument-level constraints

Selector filtering alone is insufficient when a single function can be called with dangerous arguments (e.g., `transfer(address,uint256)`). Argument-level policy decodes calldata and enforces bounds:

```
function checkAdvanceArgs(bytes calldata data) internal pure returns (bool) {
    (, uint256 amount) = abi.decode(data[4:], (uint256, uint256));
    return amount <= MAX_ADVANCE;
}
```

Parameter-level control and spending limits are enforced alongside function-level permissions.

5. Simulation Gate on Arbitrum Forks

Before execution, the agent simulates the proposed call against a forked Arbitrum state. Simulation detects calldata errors and produces a state diff that can be compared against expected bounds.

5.1 Fork simulation

Foundry fork testing runs against real chain state without deploying to a live network, using `vm.createFork`, `vm.selectFork`, and `vm.rollFork`. The agent runtime invokes the same simulation against an Arbitrum fork so that the state diff reflects the actual contract versions deployed on Arbitrum One.

```
function test_simulatePause_onArbitrum() public {
    vm.createSelectFork(vm.envString("ARBITRUM_RPC_URL"));
    uint256 before = spool.totalAssets();
    vm.prank(agent);
    spool.pause();
    assertTrue(spool.paused());
```

```
    assertEq(spool.totalAssets(), before);
}
```

The runtime pins the fork URL to the deployment chain recorded in the policy engine. Simulating against a different chain produces misleading diffs.

5.2 State-diff bounds

Action	Expected state diff	Rejection condition
Pause	Paused flag = true; no balance changes	Any balance change
Reprice	Spread parameter changes within bounds	Spread outside [min,max]
Notify	Event emitted; no storage change	Unexpected storage write
Advance	Outstanding increases by amount	Amount ≠ intended; recipient unexpected

6. Human Escalation and Multi-Sig Design

High-impact moves escalate to human multi-sig. The escalation boundary is defined by objective, on-chain criteria rather than agent judgment.

Criterion	Escalation trigger	Rationale
Value	Call value above threshold	Large transfers warrant human review
Irreversibility	Action cannot be undone	Errors are permanent
Scope	Action affects protocol-wide parameter	Systemic impact
Novelty	Selector not previously exercised	Unfamiliar code path

The multi-sig is a standard threshold scheme (m-of-n) with signers using hardware wallets or MPC. The policy engine enforces that escalated actions are only executable by the multi-sig address, so the agent cannot bypass escalation even if compromised.

7. Timeboost and Latency-Sensitive Escalation

Arbitrum Timeboost provides an optional express lane that grants a controller priority inclusion for a short window. In the default configuration, express-lane transactions are processed immediately while others face a ~200ms delay. The Agency uses Timeboost for risk-reducing actions that should not queue behind routine traffic.

Action	Timeboost use	Rationale
Pause	Recommended	Risk-reducing; latency matters
Reprice	Recommended	Time-sensitive; spread must track score
Notify	Optional	Event-only; latency less critical
Advance	Not used	Irreversible; waits for parent-chain safe finality

Independent empirical analysis of Timeboost (April–July 2025, 11.5 million express-lane transactions) found that express-lane control is highly centralized, with two entities winning more than 90% of auctions, and that auction competition declined over time. The Agency therefore treats Timeboost as an optimization for risk-reducing actions only, never as a security dependency, and never for value-moving actions. If the express lane is unavailable or unaffordable, pause and reprice still execute on the normal path with the standard ~200ms delay.

8. Key Separation and Off-Model Credentials

The design principle "keys and limits stay off-model" means the AI model never has access to signing keys, RPC credentials, or policy parameters. Three separations are enforced architecturally:

1. **Model context separation:** the model receives observations and outputs proposals; it does not receive secrets.
2. **Signing separation:** the signer is a separate service (HSM, MPC, or hardware wallet) that accepts only policy-approved calldata.

3. **Policy separation:** policy parameters are stored on-chain on Arbitrum and changed only through governance (timelocked, with the version hash anchored to Ethereum L1), not by the agent.

For cases where policy confidentiality matters, off-chain policy evaluation with an on-chain proof is available: the ruleset remains secret, the on-chain guard verifies the proof, and enforcement remains on-chain.

9. Agent Runtime and Calldata Construction

The agent runtime constructs calldata for allowlisted actions. Construction is typed and schema-validated before simulation.

```
type ProposedAction = {
  target: `0x${string}`;
  selector: `0x${string}`;
  data: `0x${string}`;
  value: bigint;
};

function buildAdvance(orderId: bigint, amount: bigint): ProposedAction {
  return {
    target: SPOOL_ADDRESS,
    selector: spoolInterface.getSighash("fundOrder"),
    data: encodeFunctionData({ abi: spoolAbi, functionName: "fundOrder", args: [orderId, amoun
    value: 0n,
  };
}
```

TypeScript is the orchestration language; a typed builder prevents the class of errors where calldata is constructed with the wrong ABI or argument order.

10. Sequencer Failure and L2 Resilience

Arbitrum's sequencer is currently centralized. If the sequencer stalls, agent actions cannot be included in the normal path. This is distinct from oracle failure and is handled separately.

- **Detection:** the agent runtime monitors Arbitrum block height; a stall beyond 30 seconds (roughly 120 missed blocks at ~250ms block time) triggers degraded mode.
- **Containment:** the agent stops proposing value-moving actions; risk-reducing actions already submitted remain valid.
- **Recovery:** critical pause actions are force-included via the Arbitrum delayed inbox on L1.
- **Policy:** the policy engine records the maximum acceptable staleness of sequencer confirmation; beyond it, escalation triggers automatically to the multi-sig, which acts via the delayed inbox.

The Agency distinguishes sequencer soft confirmation from parent-chain safe finality. A pause executed under soft confirmation is a valid risk-reducing action even if the batch has not yet reached L1, because the pause cannot cause loss by itself. An advance, by contrast, waits for parent-chain safe confirmation (~12–15 minutes) before being treated as final.

11. Security Analysis and Failure Modes

Failure	Detection	Containment
Agent submits disallowed selector	Policy engine `evaluate` returns false	Transaction reverts before target call
Agent submits allowed selector with malicious args	Argument-level policy; simulation state diff	Revert or escalation
Simulation passes but execution diverges	Post-execution invariant check	Revert if invariant broken
Multi-sig signer collusion	Threshold policy; independent signers	Requires m-of-n compromise
Policy misconfiguration	Governance review; timelock anchored to L1	Timelock allows reversal
Sequencer stall	Block-height stall detection	Degraded mode; force-include pause via L1 delayed inbox

12. Testing Strategy

- **Policy unit tests:** every allowlist and argument bound has a positive and negative test.
- **Simulation fork tests:** run the agent's proposed calldata against Arbitrum forked state and assert the expected state diff.
- **Escalation tests:** verify that high-value calls route to multi-sig and cannot be executed by the agent address.
- **Invariant tests:** fuzz sequences of allowed actions and assert system invariants hold.
- **Sequencer-failure drill:** simulate a stalled sequencer and verify that degraded mode triggers and the delayed-inbox path works for a pause.

Foundry's four testing types—unit, integration, fork, and staging—map directly onto this strategy, with the fork suite pinned to Arbitrum One.

13. Governance and Policy Upgrades

Policy changes are high-impact and are governed with a timelock anchored to Ethereum L1. The timelock gives users time to exit if a policy change is hostile, and gives defenders time to react to a compromised governance key.

- **Timelock delay:** sufficient for the protocol's risk profile (48–72 hours).
- **Policy versioning:** each policy set has a version hash; consumers verify which policy was active at execution time.
- **Emergency pause:** a separate, narrower mechanism on Arbitrum that can pause but not unpause or upgrade.
- **L1 anchor:** the timelock and policy version root live on Ethereum, so policy changes inherit L1 finality even though execution happens on Arbitrum.

> **Disclaimer.** This white paper is for informational purposes and does not constitute financial, legal, or investment advice. Digital assets and DeFi protocols carry significant risk, including total loss of capital.

© 2025 Ukiyo Network. Agency for Smart Contracts technical white paper, version 3.0.

Execution environment: Arbitrum One. Governance anchor: Ethereum L1.