Authority is a Continuous System

Nicola Gallo

8 December 2025

1 Motivation

In prior work [1], we formally showed that Proof-of-Possession (PoP)–based authorization models admit confused deputy conditions. Because authority is exercised solely through possession of a transferable artifact, deputies may reinterpret privileges outside the causal intent of the originator.

The same work showed that the Provenance Identity Continuity (PIC) model eliminates the confused deputy entirely. By enforcing monotonic authority restriction and origin-bounded execution, PIC makes the confused deputy condition non-formulable: no privilege can be exercised unless it was present at the execution origin and causally propagated through a verifiable chain.

This leads to a fundamental insight: authority is not a transferable object, but a continuity property of execution. Once this is understood—against decades of research in access control—it becomes clear that many existing models have incorrectly treated authority as a form of possession.

This paper elevates the PIC model to a more abstract and general form, analyzing authority as a continuous system and formalizing its implications for trust, governance, concurrency, and zero-trust architectures.

2 Authority Continuity Principle

Let P be a finite set of principals, O a set of operations, and R a set of resources. A privilege is a pair $(o, r) \in O \times R$.

Let $p_0 \in P$ be the immutable originating principal of an execution. The set $\text{ops}_0 \subseteq O \times R$ denotes the authority explicitly exercised within this execution. Importantly, ops_0 is a subset of the total authority of p_0; only this subset participates in execution continuity.

In prior work [1], execution was intentionally modeled as:

$$\pi = \langle (p_0, \mathrm{ops}_0), (p_1, \mathrm{ops}_1), \ldots, (p_n, \mathrm{ops}_n) \rangle, \quad \mathrm{ops}_{i+1} \subseteq \mathrm{ops}_i,$$

in order to demonstrate the elimination of the confused deputy under monotonic restriction.

For a more general and abstract treatment, we model execution as a continuous flow of authority rather than a reassignment between principals.

The originating principal of an execution is immutable.

Execution is represented as a chain of authority states:

$$\pi = \langle \alpha_0, \alpha_1, \ldots, \alpha_n \rangle, \quad \alpha_i = (p_0, \mathrm{ops}_i, e_{i-1}, e_i),$$

where $e_i \in P$ is the executor performing hop i.

Monotonicity holds for all i:

$$\mathrm{ops}_{i+1} \subseteq \mathrm{ops}_i.$$

3 Guardrails

Authority continuity is enforced by invariant constraints evaluated at every execution step i. Guardrails validate continuity; they never grant authority.

3.1 Origin Guardrail

$$\forall i : \; p_0^{(i)} = p_0^{(0)}.$$

3.2 Operational Guardrail

$$\forall i : \; \mathrm{ops}_{i+1} \subseteq \mathrm{ops}_i.$$

3.3 Executor Guardrail

$$\forall i : \; \mathrm{ExecCont}(p_0, e_i, e_{i+1}, \mathrm{ops}_{i+1}).$$

3.4 Temporal Guardrail

Temporal validity is defined by a predicate, not by simple set containment.

Let τ_i be the current time, included in the execution context. Let T_i denote a temporal constraint associated with state α_i.

$$\text{TimeValid}(T_i, \tau_i) \in \{\mathsf{true}, \mathsf{false}\}.$$

The temporal constraint T_i may represent:

- an absolute time range $[t_{\min}, t_{\max}]$ such that $\text{TimeValid}(T_i, \tau_i)$ holds iff $\tau_i \in [t_{\min}, t_{\max}]$;
- a relative duration from an origin time t_0 (e.g. valid for Δt after t_0), such that validity holds iff $\tau_i - t_0 \leq \Delta t$;
- an unbounded (infinite) interval, in which case $\text{TimeValid}(T_i, \tau_i)$ is always true.

If no temporal constraint is specified, execution is temporally unbounded.

3.5 Contextual Guardrail

Context is validated by predicate, not merely by set inclusion.

Let C_i be the execution context associated with α_i and X_i external signals (including time and environmental conditions).

$$\text{CtxValid}(C_i, X_i) \in \{\mathsf{true}, \mathsf{false}\}.$$

Simple containment $C_{i+1} \subseteq C_i$ is a sufficient but not necessary condition. Context validity may depend on policies, environment, or other external constraints.

4 Executor Continuity Predicate

[Executor Continuity] The predicate

$$\text{ExecCont}(p_0, e_i, e_{i+1}, \text{ops})$$

holds iff executor e_{i+1} is a causally valid continuation of execution from e_i under origin p_0 and authority ops.

Executor continuity may depend on implementation-specific predicates, including but not limited to:

- proof that e_{i+1} is the executor it claims to be,
- proof of required execution attributes or environment,
- proof of contextual binding (e.g. request provenance),
- proof that execution proceeds within authority ops.

Crucially, ExecCont:

- does not grant authority,
- does not expand authority,
- does not preserve authority,
- does not substitute authority continuity.

It merely validates that execution at hop $i+1$ is causally derived from execution at hop i and is therefore eligible to participate in authority continuity.

5 Authority Succession

Authority evolves by applying all guardrails atomically.

We enrich authority states as:

$$\alpha_i = (p_0, \mathrm{ops}_i, e_{i-1}, e_i, T_i, C_i).$$

Given proposed inputs

$$(e_{i+1}, \mathrm{ops}_{i+1}, T_{i+1}, C_{i+1}),$$

we construct the candidate successor:

$$\alpha_{i+1} = (p_0, \mathrm{ops}_{i+1}, e_i, e_{i+1}, T_{i+1}, C_{i+1}).$$

We define the combined guardrail predicate:

$$\begin{aligned}\mathrm{Guard}(\alpha_i, \alpha_{i+1}) \equiv\ & \big(p_0^{(i+1)} = p_0^{(i)}\big) \wedge \big(\mathrm{ops}_{i+1} \subseteq \mathrm{ops}_i\big) \wedge \\ & \mathrm{ExecCont}\big(p_0, e_i, e_{i+1}, \mathrm{ops}_{i+1}\big) \wedge \\ & \mathrm{TimeValid}(T_{i+1}, \tau_{i+1}) \wedge \mathrm{CtxValid}(C_{i+1}, X_{i+1}).\end{aligned}$$

The succession function is:

$$\mathsf{Next}(\alpha_i,\, e_{i+1},\, \mathrm{ops}_{i+1},\, T_{i+1},\, C_{i+1}) = \begin{cases} \alpha_{i+1} & \text{if } \mathrm{Guard}(\alpha_i, \alpha_{i+1}) = \mathsf{true}, \\ \mathsf{halt} & \text{otherwise.} \end{cases}$$

Succession:

- constructs a candidate α_{i+1},
- applies all guardrails as a single predicate,
- returns α_{i+1} only if continuity is preserved,
- otherwise halts execution at α_i.

Succession preserves continuity and never introduces new authority.

6 Transactional Structure of Authority Continuity

Authority continuity admits both sequential and parallel evolution.

6.1 Sequential Continuation

Repeated application of Next yields linear chains:

$$\alpha_0 \to \alpha_1 \to \cdots \to \alpha_n.$$

Failure of a single transition halts execution.

Sequential composition is purely structural: the existence of a syntactic sequence $\alpha_i \to \alpha_{i+1}$ does not, by itself, guarantee that execution is authorized to proceed. Each transition in the chain is realized only when the corresponding call to Next returns α_{i+1} (i.e. when Guard holds and any governance decisions allow it); otherwise execution halts at α_i.

6.2 Parallel Continuation (Fork)

Given a state α_i, multiple successors may be generated:

$$\alpha_{i+1}^{(1)}, \ldots, \alpha_{i+1}^{(k)},$$

each constructed via Next with different parameters.

Every branch must independently satisfy all guardrails. Forking does not duplicate or expand authority: each branch maintains monotonicity relative to its own execution line.

The existence of multiple successor candidates does not, by itself, authorize execution along any branch. For each branch j, the transition $\alpha_i \to \alpha_{i+1}^{(j)}$ becomes effective only if the corresponding Next call returns $\alpha_{i+1}^{(j)}$ and any governance decisions permit that branch to proceed; otherwise the branch halts at α_i.

6.3 Join (Convergent Continuation)

Let $\alpha_i^{(1)}, \ldots, \alpha_i^{(k)}$ be states that share the same origin p_0. A join function produces a new candidate state:

$$\mathsf{Join}(\alpha_i^{(1)}, \ldots, \alpha_i^{(k)}) \to \alpha_i'.$$

Let:

$$\mathrm{ops}' \subseteq \bigcap_j \mathrm{ops}_i^{(j)}, \quad T' \subseteq \bigcap_j T_i^{(j)}, \quad C' \subseteq \bigcap_j C_i^{(j)}.$$

Join is thus a transformation that produces α_i' with authority, time, and context no greater than what is already present in the input branches. Join by itself does not imply authorization to execute. The resulting state α_i' must be treated as a candidate and, for any subsequent step $\alpha_i' \to \alpha_{i+1}'$, must again be passed through:

- continuity guardrails via Next and Guard,
- any applicable governance decisions via PDP.

Authority after join can never exceed the authority available before join.

7 Trust, Possession, and Execution

Authority continuity cleanly separates execution, trust, and possession.

7.1 Executor Verification

At each hop exactly one executor performs the concrete action. The executor is the only element that must demonstrate anything: that it is in fact the executor it claims to be.

This demonstration binds execution to reality (identity, environment, attributes) but does not confer additional authority and does not alter authority continuity.

7.2 Possession as Binding, Not Authority

Proof of Possession (PoP) has a purely binding role:

- it associates identity, attributes, environment, or context with execution,
- it provides *untrusted evidence* that may be evaluated by ExecCont, TimeValid, or CtxValid,

but it does *not*:

- provide trusted evidence for continuity, temporal, or contextual validation,
- create authority,
- preserve authority,
- propagate authority,
- justify authority expansion.

Possession is orthogonal to continuity.

7.3 Trust as Continuity

Trust does not arise from possession of an artifact. Trust emerges from verified causal continuity.

An executor is considered trustworthy if and only if its execution constitutes a valid continuation from the immutable origin p_0 under all guardrails and any applicable governance decisions.

Trust may decrease or be revoked over time by changing policies, but such revocation does not introduce discontinuity: it halts or restricts the evolution of authority.

8 Governance as Policy-Constrained Continuity

Continuity defines what is structurally possible. Governance defines what is currently permitted.

Let Policy be the set of governance rules (which may change over time) and X_i the external context (network state, risk signals, operational conditions, etc.).

We define the policy decision predicate:

$$\mathrm{PDP}(\alpha_i, \alpha_{i+1}, X_i, \mathrm{Policy}) \in \{\mathsf{allow}, \mathsf{deny}\}.$$

A transition $\alpha_i \to \alpha_{i+1}$ may occur if and only if:

$$\mathrm{Guard}(\alpha_i, \alpha_{i+1}) \ \wedge \ \mathrm{PDP}(\alpha_i, \alpha_{i+1}, X_i, \mathrm{Policy}) = \mathsf{allow}.$$

Governance:

- may deny transitions that are structurally valid,
- may further restrict operations, time, and context,
- may terminate execution in response to external conditions,

but it cannot:

- introduce new authority,
- expand ops beyond what continuity allows,
- bypass guardrails.

In continuous-authority systems, governance is therefore an inherently authority-reducing layer.

9 Zero-Trust Incompatibility of PoP-Based Propagation

[Zero-Trust Compatibility] A system is compatible with the zero-trust model if every execution advance is produced exclusively by the succession function Next and satisfies all guardrails, possibly subject to governance decisions.

[PoP-Based Propagation] A system is *PoP-based* if there exists an artifact a such that, for some contexts, authority at a subsequent hop can be (re)materialized solely on the basis of possession of a, independently of the current authority state.

No authorization system based on Proof of Possession is compatible with the zero-trust model under the authority continuity assumption.

Proof. PoP-based propagation allows authority to be reconstructed in contexts that do not causally derive from the current state α_i. Authority continuity instead requires monotonic, causally derived succession constrained by Guard and, optionally, by PDP.

Therefore, there exist execution steps enabled solely by possession of a that cannot be represented as $\mathsf{Next}(\alpha_i, \dots)$ with Guard $= \mathsf{true}$. Such steps violate zero-trust compatibility as defined above.

> *Systems that derive authority from Proof of Possession are not zero-trust systems by construction.*

10 Conclusion

Authority is not an artifact, a credential, or a possession. It is a continuous, causally constrained property of execution rooted in an immutable origin.

By modeling authority as continuity:

- authority propagation becomes intrinsically monotonic,
- concurrency is handled via fork and join under continuity,
- governance is expressed as a policy layer that only restricts,
- the zero-trust model emerges as a native property of the system.

Systems that treat authority as possession cannot, by construction, achieve these properties. Authority continuity instead provides a unified foundation for re-thinking security, identity, and access control in distributed systems.

Acknowledgments

The author used automated language assistance tools exclusively for grammar and phrasing. All conceptual contributions and formalizations are solely the author's.

References

1. N. Gallo. *Authority Propagation Models: PoP vs PoC and the Confused Deputy Problem.* Zenodo, 2025.
2. N. Gallo. *PIC Model—Provenance Identity Continuity for Distributed Execution Systems.* Zenodo, 2025.

3. N. Hardy. *The Confused Deputy.* Operating Systems Review, 1988.

4. J. Kindervag et al. *Zero Trust Architecture.* NIST Special Publication 800-207, 2020.

5. Google. *BeyondCorp: A New Approach to Enterprise Security.* ;login:, 2014.

6. B. Lampson. *Protection.* ACM Operating Systems Review, 1974.