

**TDI 2026**

April 20-21, 2026 · Verona, Italy

**Provenance Identity Continuity (PIC):** *Secure Authority Propagation from Human and Non-Human Origins Across Trust Boundaries*

21 April 2026

**Nicola Gallo**  
*Co-founder, Nitro Agility S.r.l.*

**Antonio Radesca**  
*Co-founder, Nitro Agility S.r.l.*





# From Possession to Execution

- A **token** can **prove**: “I hold valid authority now.”
- It **cannot prove**: “this step is still **part** of the **same authorized execution**.”
- That **gap** is **not a policy gap**.

It is a **model gap**

This talk explains:

- **why possession** is **local**
- **why execution** is **temporal-causal**
- and **why execution continuity** must become **part** of the **security model**

**Not a better token.**

A **different primitive.**



# What This Talk Is Not About

It is **not about**:

- Centralization vs Decentralization
- OAuth vs Capabilities

It is **about**:

- security models
- distributed execution
- authority propagation through execution

PIC changes the **primitive**:

- **Proof of Relationship (PoR)** makes execution membership representable
- **Proof of Continuity (PoC)** makes valid continuation verifiable

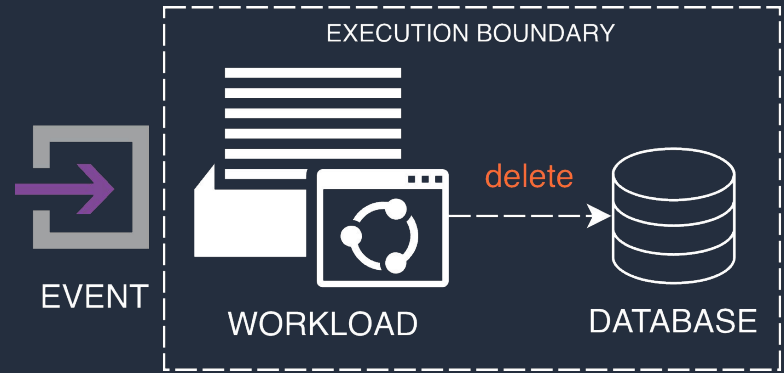
From **authority-as-possession**  
to **authority-as-continuity**

# A Workload Is Autonomous Within Its Execution Boundary

## First Precondition of the Execution Model

It can act at any time — triggered by a request, an event, or even by itself.

And once it starts executing, it can do anything within its execution boundaries — across its resources, devices, and connected systems.

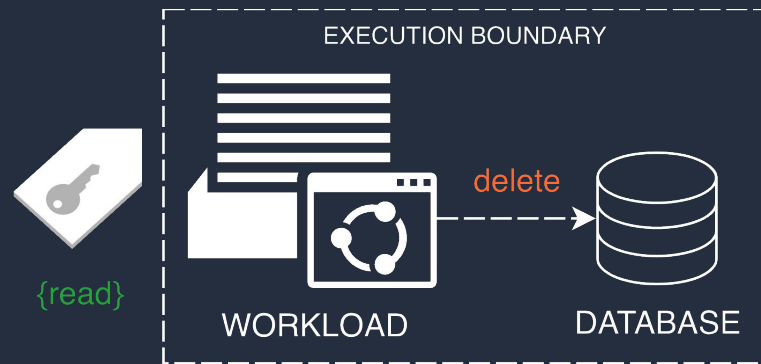


# Execution Can Deviate from Intent

## Second Precondition of the Execution Model

Even with a **valid request** and **valid authority**, it may do something completely different within its execution boundaries.

Whether it's a **bug** or **malicious** behavior **doesn't matter**.



# What Does Trust Mean in an Execution Model?

## Trust Assumptions in an Execution Model

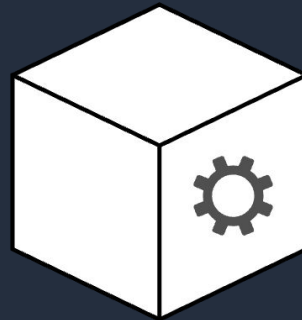
The key point is:

**We cannot control what happens inside a workload.**

When we say we “trust” a workload, we usually mean:

- We trust **who operates it**
- We trust the **organization behind it**
- We trust their **engineering practices**
- We trust their **security measures**
- We trust the **software behaves as intended**

 [www.acmecorp.com](http://www.acmecorp.com)



Trust me, it is me and I know what I am doing. I take responsibility.

# We already trust external providers

We already **give** our **data** to **systems** we **don't control**.

We **trust** them to **behave correctly**.

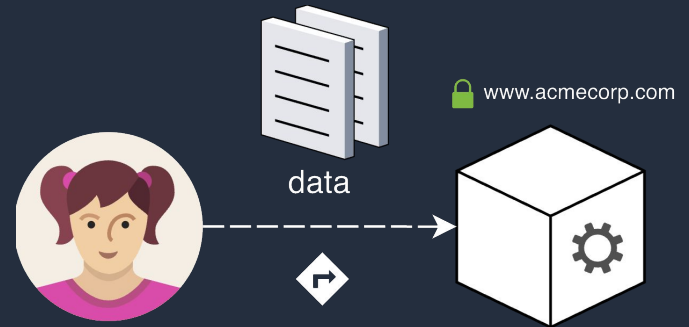
We **trust** their **security practices**.

But **that's not security** — **that's trust**.

**AI agents** don't introduce a new problem.

They **make** an **existing assumption visible**.

What's **missing** is a **model** to **reason** about **authority** and **execution**.



# What Does Trust Mean for AI Agents?

## The Problem Is the Framing

When we say we **trust AI agents**

- We trust the **organization behind them**
- We trust their **engineering** and **safeguards**

But **AI agents** are **non-deterministic**

- They can produce **unpredictable outcomes**
- They **cannot guarantee correctness**

This is not a reliability problem

A **Trust Problem** not a **Security Model Problem**



www.acmecorp.com



Sure that's running on my premises but I can't guarantee it. It is non-deterministic, don't blame me.

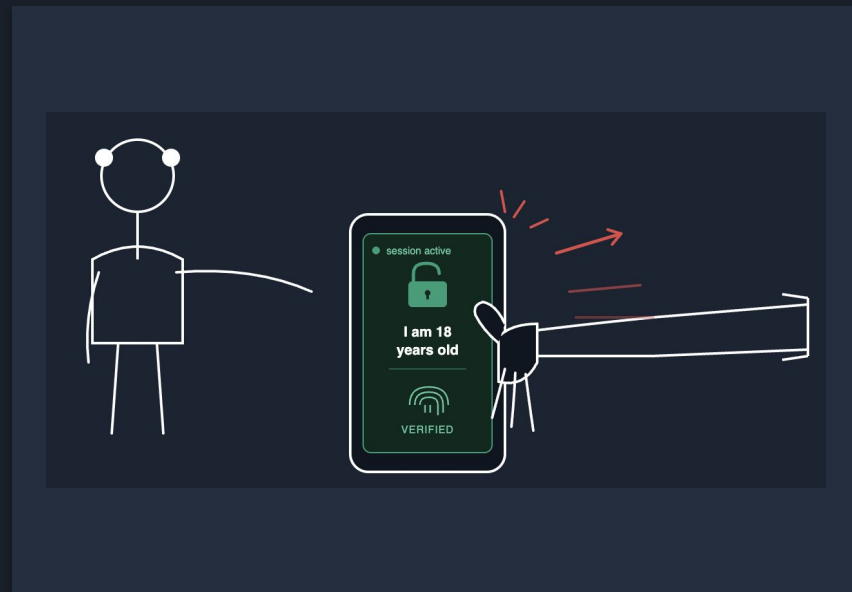
# Trust Begins Before the Security Model

## Physical Preconditions of Authentication and Intent

When we talk about trust, we mean one thing:  
the process was initiated legitimately.

- If Alice unlocks her phone with her fingerprint and someone snatches it from her hand while the session is open, that's not a software problem. It's physics.
- If Alice asks Bob to act on her behalf using his mobile, that's not something cryptography can prevent. It's physics.

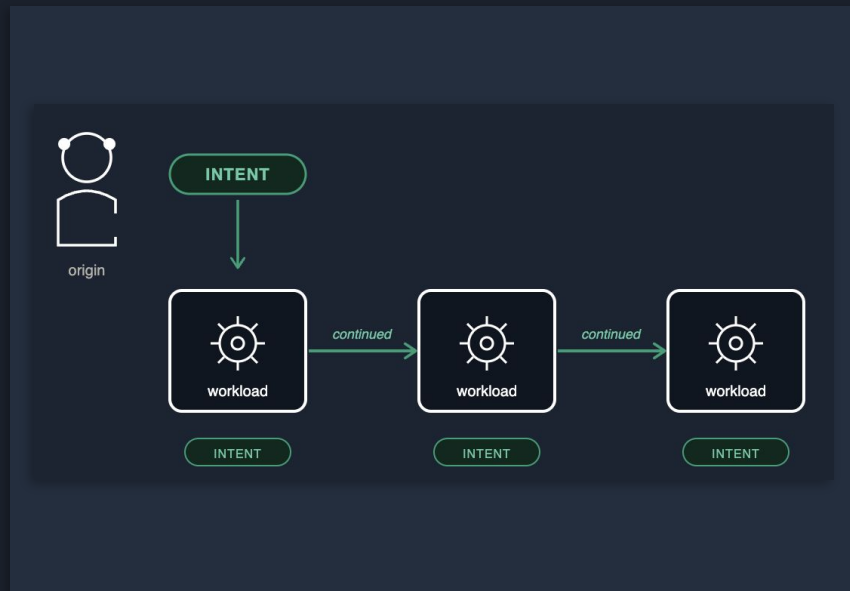
Authentication and intent are preconditions, not guarantees.  
No amount of checks downstream can undo a compromised origin.



# Machines don't prove attributes. They continue intent

## Physical Preconditions of Authentication and Intent

It does not need to choose which identity to present or which attestations it holds — it only needs to continue.



# Trust Begins Before the Security Model

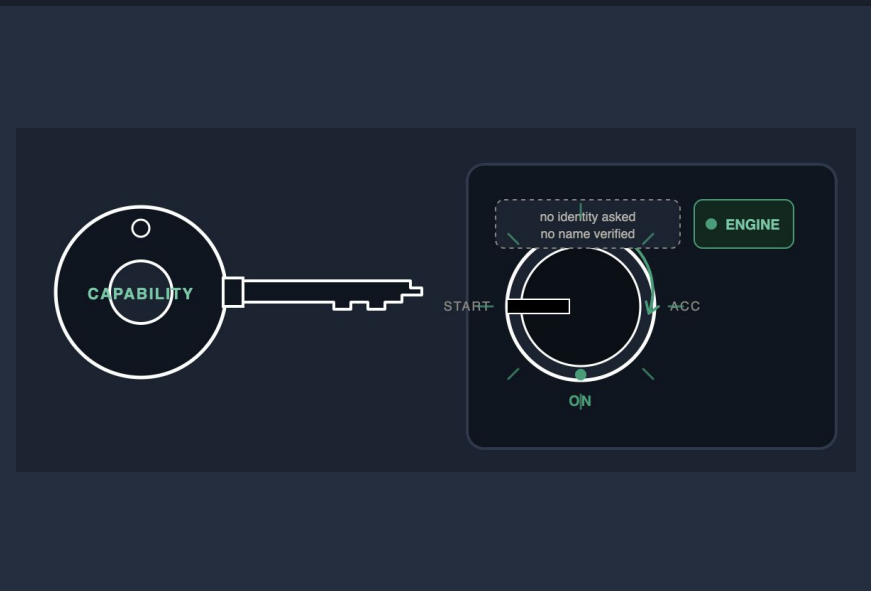
## Physical Preconditions of Authentication and Intent

If **privacy matters**, once the **origin is assumed** correct, the **model** can take over:

- **authority becomes anonymous capability**

The **physical problem** at the **origin** cannot be mitigated by software.

Given that, **everything else** becomes **execution** and **authority propagation**, something we can reason about.





# Trust & Security Model

- **Trust Model: Who/what is trusted, for what, under which conditions.**  
*Trust anchors, boundaries, assumptions (e.g., trusted IdP/AS, accepted issuers, audiences, delegation).*
- **Security Model: How the system is protected given those assumptions.**  
*Mechanisms, policies, enforcement (e.g., OAuth 2.1/OIDC, Capabilities, PKCE, DPoP, mTLS, RBAC/ABAC).*

# What about the Security Model?

Workloads are connected to each other. Each has an execution boundary.

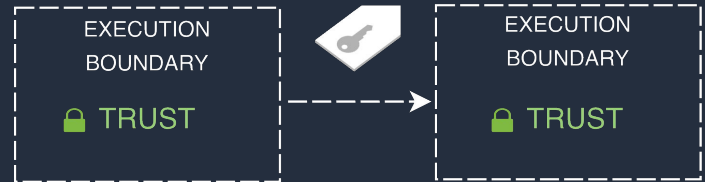
- Internal behavior cannot be controlled
- Trust is assumed (Hardware, OS, Configuration, Security practices, Risk management)

A security model connects the execution boundaries

- Between workloads
- Through their interactions

A security model = constraints on communication

Not what a workload does  
but what it can ask others to do



# What about Security Models?

Communication is a chain of delegated executions

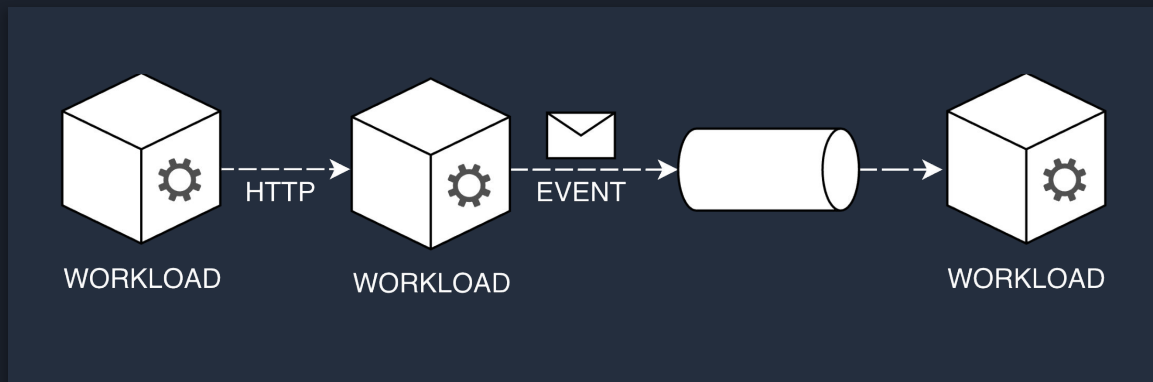
- Across **workloads**
- Across **boundaries**

It uses **different transports**

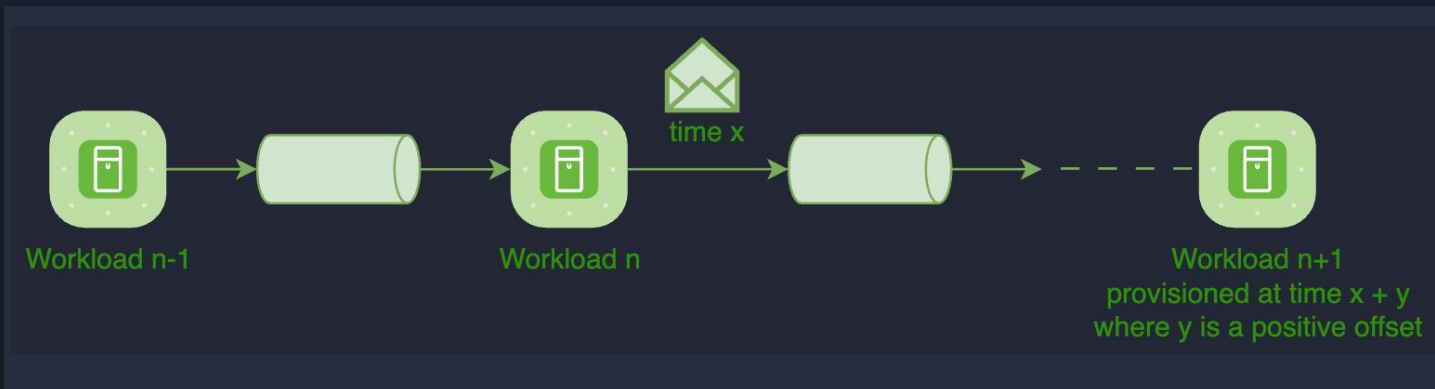
- **HTTP, gRPC**
- **Messaging, Streams**

But **execution remains the same**

**A causal chain over time**



# Canonical Execution Model



*Execution is temporal and causal, not positional.*

- Workload n+1 provisioned at time  $x + y$  ( $y > 0$ )
- Authority flows from origin — never re-created
- Every hop must prove continuity, not possession

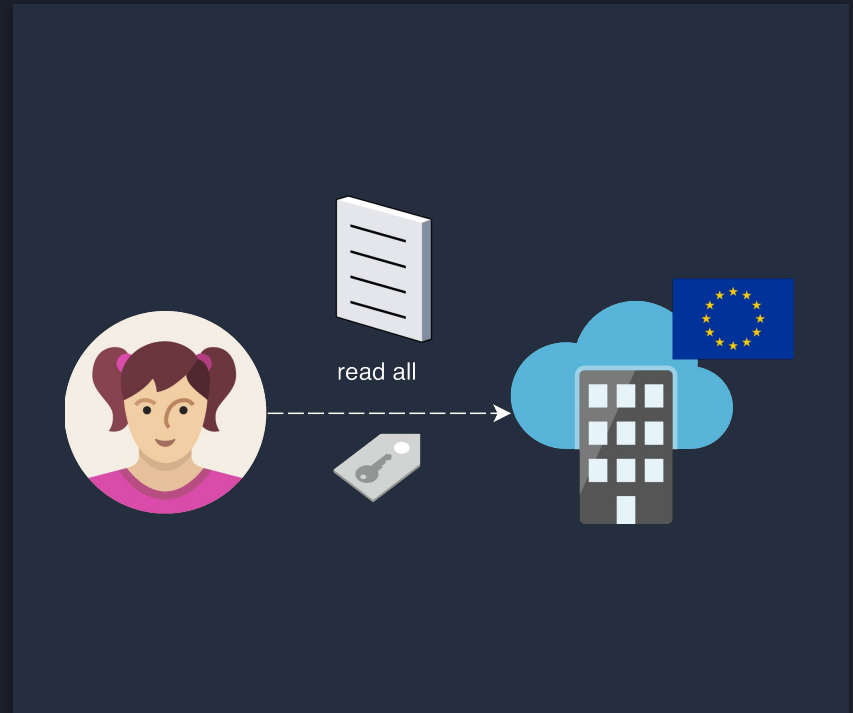
# The Alice Example

## Alice trusts the provider

- Alice connects to a saas
- She provides access to her files
- She relies on the service agreement

Backup will run under defined constraints  
(e.g. computation in EU)

Trust is assumed

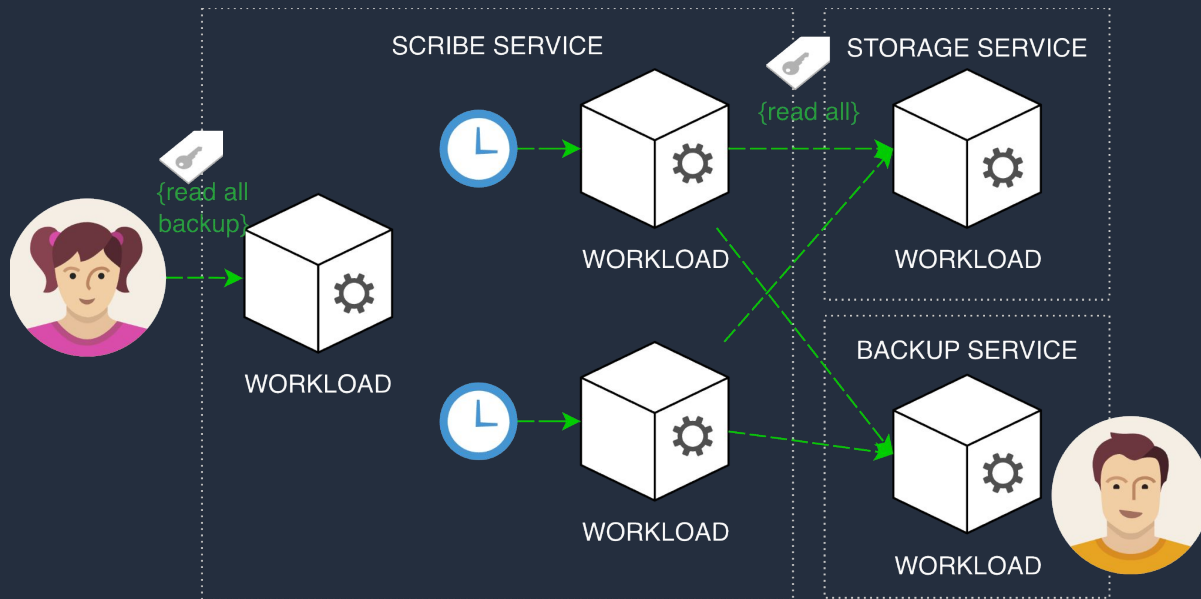


# The Alice Example

## Periodic execution

- The SaaS periodically runs backups
- It calls the storage service
- It retrieves and stores Alice's data

Execution matches the intended authority



# The Alice Example

## A new request

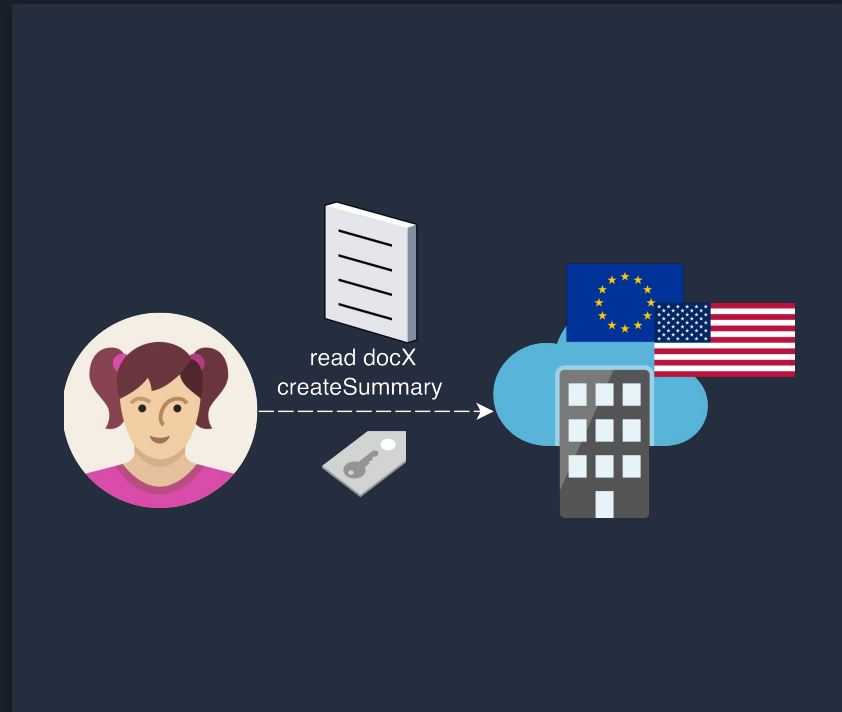
The **provider** introduces a **new feature**

## Document summarization

Alice provides a **new request**:

- Summarize DocX
- Classified as low risk
- Send summary to a collaborator

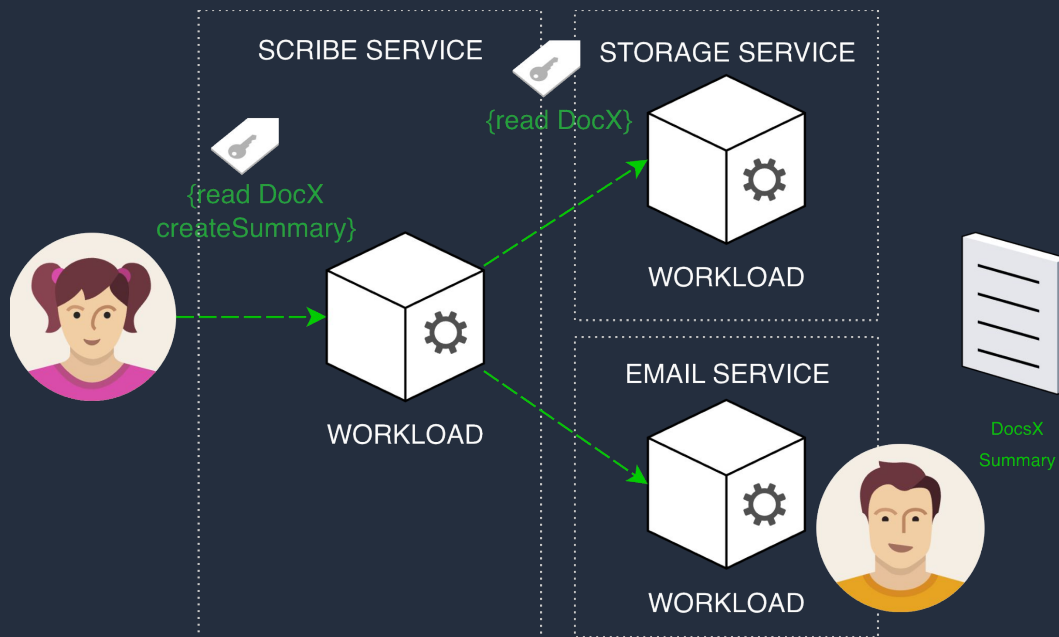
A **new, scoped intent** is defined



# The Alice Example

## Correct behavior

- The SaaS executes the summary
- Uses the scoped access
- Sends the result
- Execution matches intent

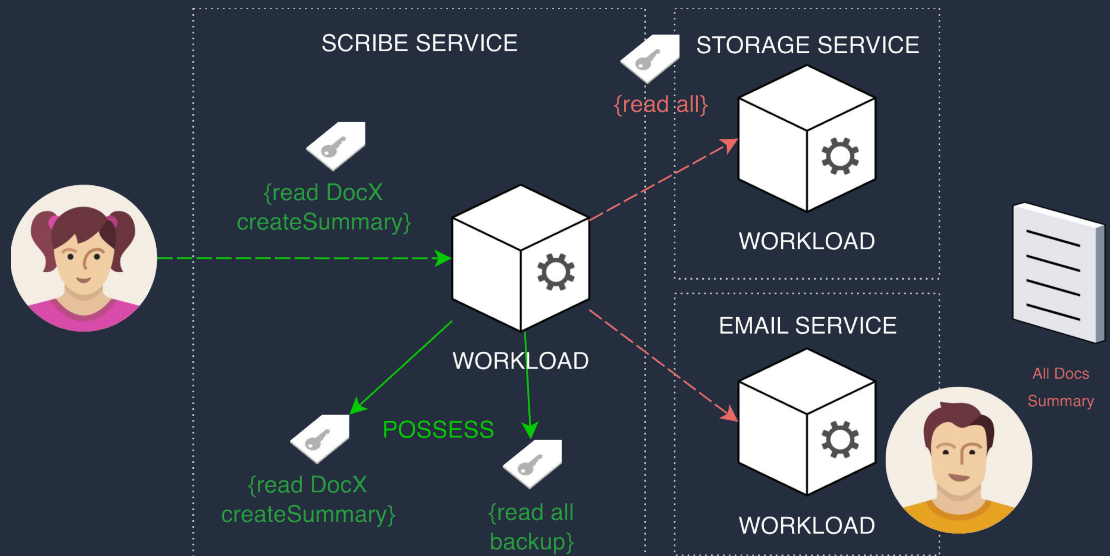


# The Alice Example

## Execution confusion

- The SaaS mixes execution contexts
- Uses the backup token instead of the summary token
- Retrieves all documents
- Generates a broad summary
- Sends it out

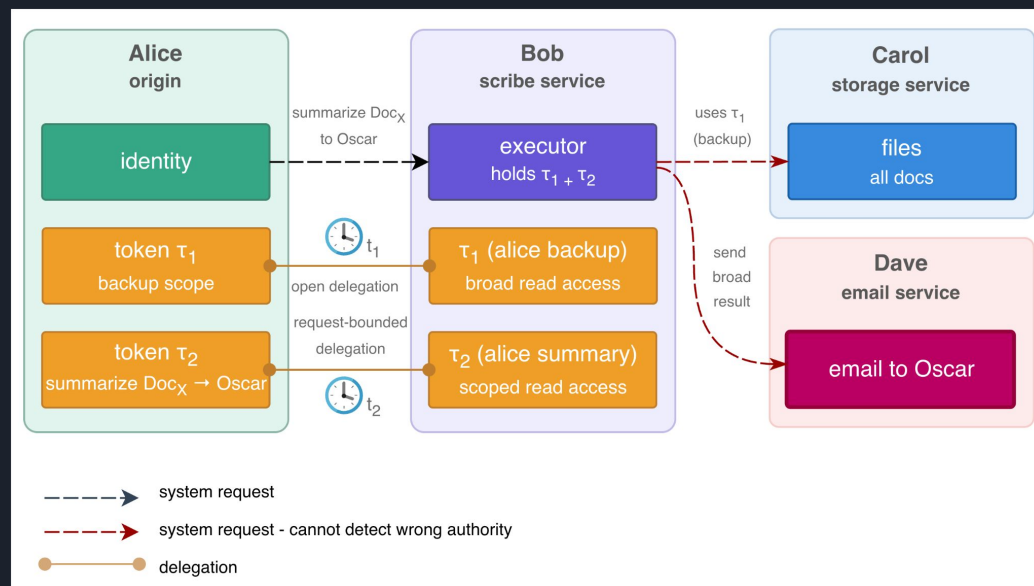
Execution does not match intent



# The Alice Example

## Confused Deputy (Revisited)

This is a **revisited confused deputy problem** where **authority is misused within the same identity**.



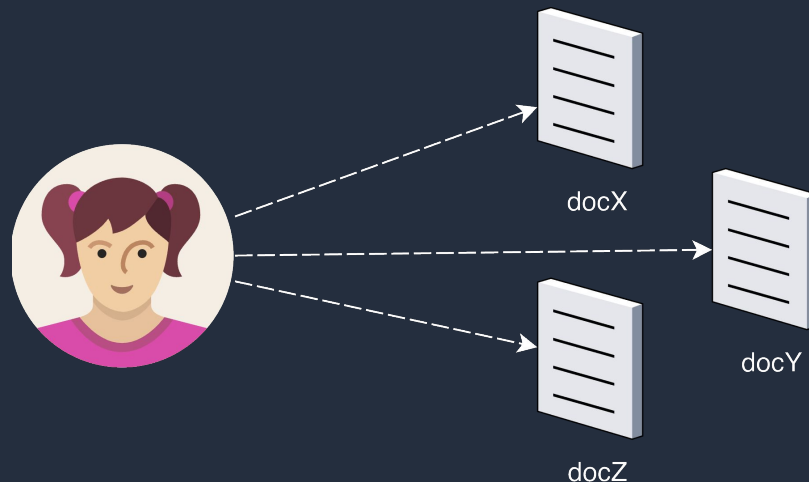
# At this point, it's clear that...

Same identity  
Same permissions

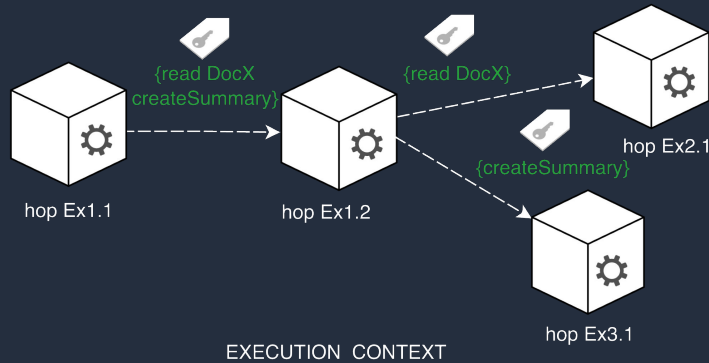
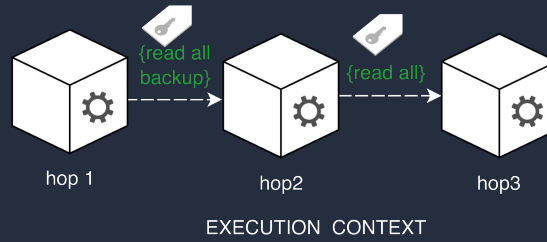
No permission expansion  
Still wrong

Execution intent was expanded  
Authority applied to the wrong execution context

Authority did not exist for this execution  
Confused deputy — within the same identity



At this point, it's clear that...



# The Execution Model

## From Proof of Possession to the Proof of Continuity

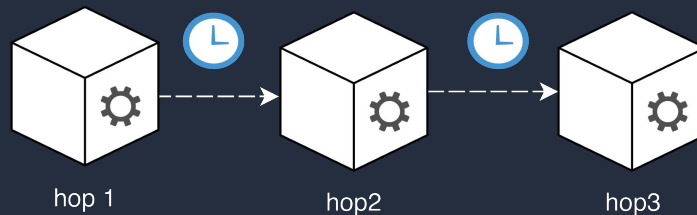
We don't need **Proof of Possession**

But we need to prove that execution is correct

- A **chain of hops**
- **Ordered in time**

A **causal chain**

$$\pi = \langle (p_0, ops_0), (p_1, ops_1), \dots, (p_n, ops_n) \rangle$$



# The Execution Model

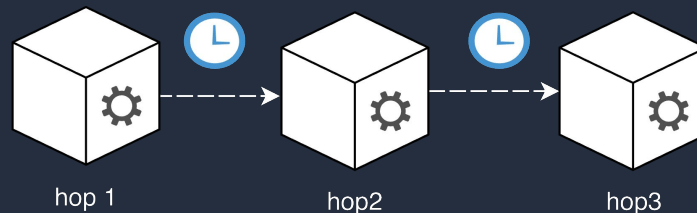
## Execution Invariants

Execution is constrained by invariants

- Bound to the origin
- No expansion of authority

Every step must respect both

Across the entire execution chain



**Definition 1** (PIC Model). A system enforces Provenance Identity Continuity if, for every execution chain  $\pi$ , the set of privileges at the final hop is bounded by the privileges at the origin:

$$ops_n \subseteq ops_0$$

and every privilege exercised at hop  $n$  is causally linked to the origin via a verifiable chain.

**Theorem 1** (PIC Safety). If the PIC Model holds, no principal can exercise a privilege not present at the origin.

*Proof.* By construction,  $ops_{i+1} \subseteq ops_i$  for all  $i$ , so  $ops_n \subseteq ops_0$ . Thus, any  $(o, r) \in ops_n$  must also be in  $ops_0$ . Therefore, no privilege can be gained beyond the origin.

# The Proof of Continuity

## Not possession, but Continuity of Execution

At **each step**, the executor **must prove**:

- Its **relation** to the **execution**
- Its link **to** the **previous step**
- Uses its **characteristics**

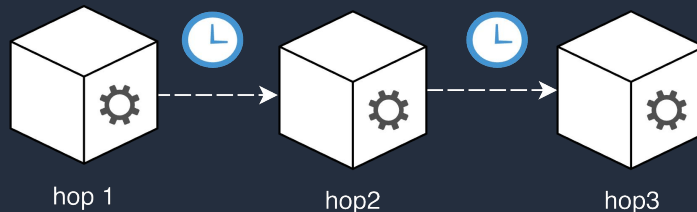
## Proof of Relationship (PoR)

And verifying invariants:

- **Bound** to the **origin**
- **No expansion** of **authority**

These **together** enable:

## Proof of Continuity (PoC)



## 4 Proof-of-Continuity (PoC / PIC)

Execution is a causal chain:

$$p_0 \rightarrow p_1 \rightarrow \dots \rightarrow p_n,$$

with  $p_0 = U$ .

Each hop transfers:

$$ops_{i+1} \subseteq ops_i.$$

Let:

$$\pi = \langle (p_0, ops_0), (p_1, ops_1), \dots, (p_n, ops_n) \rangle.$$



# Structural Elimination

## Detect

Monitor for confused deputy  
at runtime

*Traditional approach*

## Mitigate

Add scopes, bindings,  
policies to reduce risk

*Current best practice*

## Eliminate

Make the attack  
structurally inexpressible

*PIC*

**The Confused Deputy becomes non-formulable. Not prevented — inexpressible.**

# Structural Elimination

## 4.1 Authorization Rule

$(o, r)$  is authorized at the final hop iff:

$$(o, r) \in \bigcap_{i=0}^n ops_i.$$

Since the chain is monotonic decreasing:

$$\bigcap_{i=0}^n ops_i = ops_n.$$

Thus no hop may acquire authority not already present at the origin.

**Lemma 1** (Irreversible loss of authority). *If  $(o, r) \notin ops_j$  for some hop  $j$ , then  $(o, r) \notin ops_k$  for all  $k \geq j$ .*

*Proof.* By monotonicity,  $ops_k \subseteq ops_j$  for all  $k \geq j$ .

## 5 Safety Property

**Definition 3** (Origin-bounded authority). *A model enforces origin-bounded authority if every executable privilege at hop  $n$  was originally granted at hop 0.*

**Theorem 3** (PoC prevents confused deputy). *If  $ops_0 = \{(convert, r)\}$ , then  $(write, r)$  can never be authorized.*

*Proof.*  $(write, r) \notin ops_0$  and  $ops_{i+1} \subseteq ops_i$ . Thus  $(write, r) \notin ops_n$ .

## 6 Elimination of the Confused Deputy

The Confused Deputy requires an ontological assumption: that authority is a transferable or rebindable object that a principal may hold or reinterpret on behalf of another.

PIC eliminates this ontology entirely.

**Theorem 4** (Confused Deputy is non-formulable under PIC). *In any system satisfying PIC monotonicity  $ops_{i+1} \subseteq ops_i$  and origin-bounded authority  $ops_n \subseteq ops_0$ , the Confused Deputy conditions cannot be jointly satisfied.*

*Proof.* The Confused Deputy requires:

$$(o, r) \notin ops_0 \quad \text{and} \quad (o, r) \in ops_k \quad \text{for some } k > 0.$$

But PIC enforces:

$$ops_k \subseteq ops_0 \quad \forall k.$$

Thus if  $(o, r) \notin ops_0$ , it cannot appear in any  $ops_k$ . The privilege mismatch required for a Confused Deputy is impossible.

**Interpretation.** Authority in PIC is not something that a subject *has*; it is a continuity property of the execution chain. Since no hop may introduce or reinterpret authority absent at the origin, no deputy can be “confused.” The attack does not occur; it is not even definable in the model.

# The formal Proof

The screenshot shows a web browser window with the URL `zenodo.org/records/17833000`. The page content includes a link to the PIC Model and a list of files. The file `pic-model.pdf` is selected, and its content is displayed in a viewer. The document title is "Authority Propagation Models: PoP vs PoC and the Confused Deputy Problem" by Nicola Gallo, dated 1 December 2025. The document is on page 1 of 6. The visible text in the PDF includes:

**1 Model**

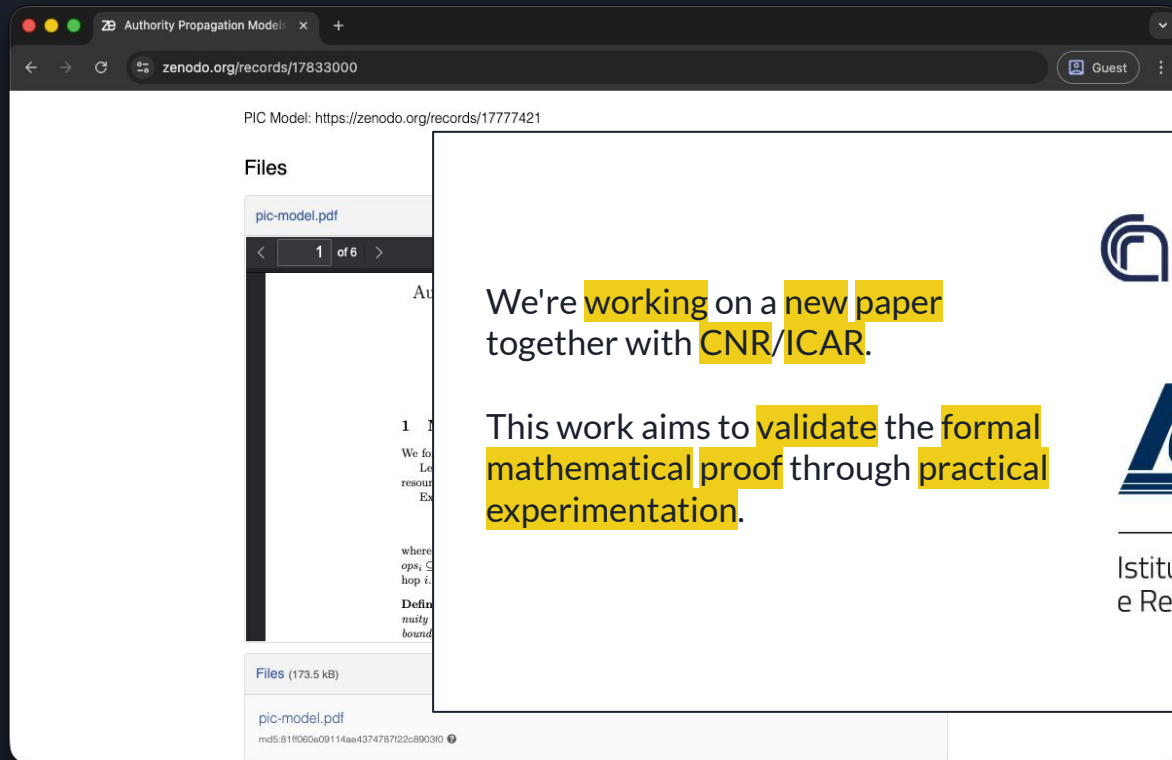
We formalize the PIC (Provenance Identity Continuity) Model as follows. Let  $P$  be a finite set of principals,  $O$  a set of operations, and  $R$  a set of resources. Define a privilege as  $(o, r) \in O \times R$ . Execution is modeled as a causal chain of hops:

$$\pi = ((p_0, ops_0), (p_1, ops_1), \dots, (p_n, ops_n))$$

where  $p_0$  is the originator and each  $ops_{i+1} \subseteq ops_i$  (monotonicity). Here  $ops_i \subseteq O \times R$  denotes the set of privileges that principal  $p_i$  may exercise at hop  $i$ .

**Definition 1** (PIC Model). A system enforces Provenance Identity Continuity if, for every execution chain  $\pi$ , the set of privileges at the final hop is bounded by the privileges at the origin:

# The Research Journey



Authority Propagation Model: x +

zenodo.org/records/17833000

PIC Model: <https://zenodo.org/records/1777421>

Files

pic-model.pdf

< 1 of 6 >

Aut

1

We fo  
Le  
resour  
Ex

where  
ops; C  
hop i.

Defin  
naily  
bound


Files (173.5 kB)


pic-model.pdf

md5:811f060e09114aa4374787f22c890310

We're working on a new paper together with CNR/ICAR.

This work aims to validate the formal mathematical proof through practical experimentation.

 Consiglio Nazionale delle Ricerche

 ICAR CNR

Istituto di Calcolo e Reti ad Alte Prestazioni

# Provenance Identity Continuity (PIC) Model



## Provenance

*The causal chain is always traceable and auditable. From Origin to end, unbroken. If it breaks, it stops.*



## Identity

*The origin is immutable. It generates authority. It cannot change throughout the chain.*



## Continuity

*Continuity is proven at every step. Proof of Continuity replaces Proof of Possession. Authority can only shrink.*



# A different Ontology

A subject creates authority by expressing intent.

After that:

**Authority must only be continued, never recreated.**

**Humans** (OIDC, VC)    **Non-humans** (SPIFFE, DID/VC)



# A New Primitive is Required

We **need** a **new primitive**.

Tokens encode possession.

We **need** a **primitive** that **encodes intent** and **execution**.



# Proof of Continuity (PoC)

To continue authority, a workload must prove:



## Causal Continuity

Execution is a valid continuation of its predecessor.



## Origin Invariance

The origin ( $p_0$ ) is immutable throughout the chain.



## Monotonic Authority Restriction

Authority is monotonically non-increasing.  $ops_{i+1} \subseteq ops_i$  — authority never expands.

**Proof of Continuity (PoC)** is built from **Proof of Relationship (PoP)**.  
**Proof of Possession** is one mechanism, not the primitive.



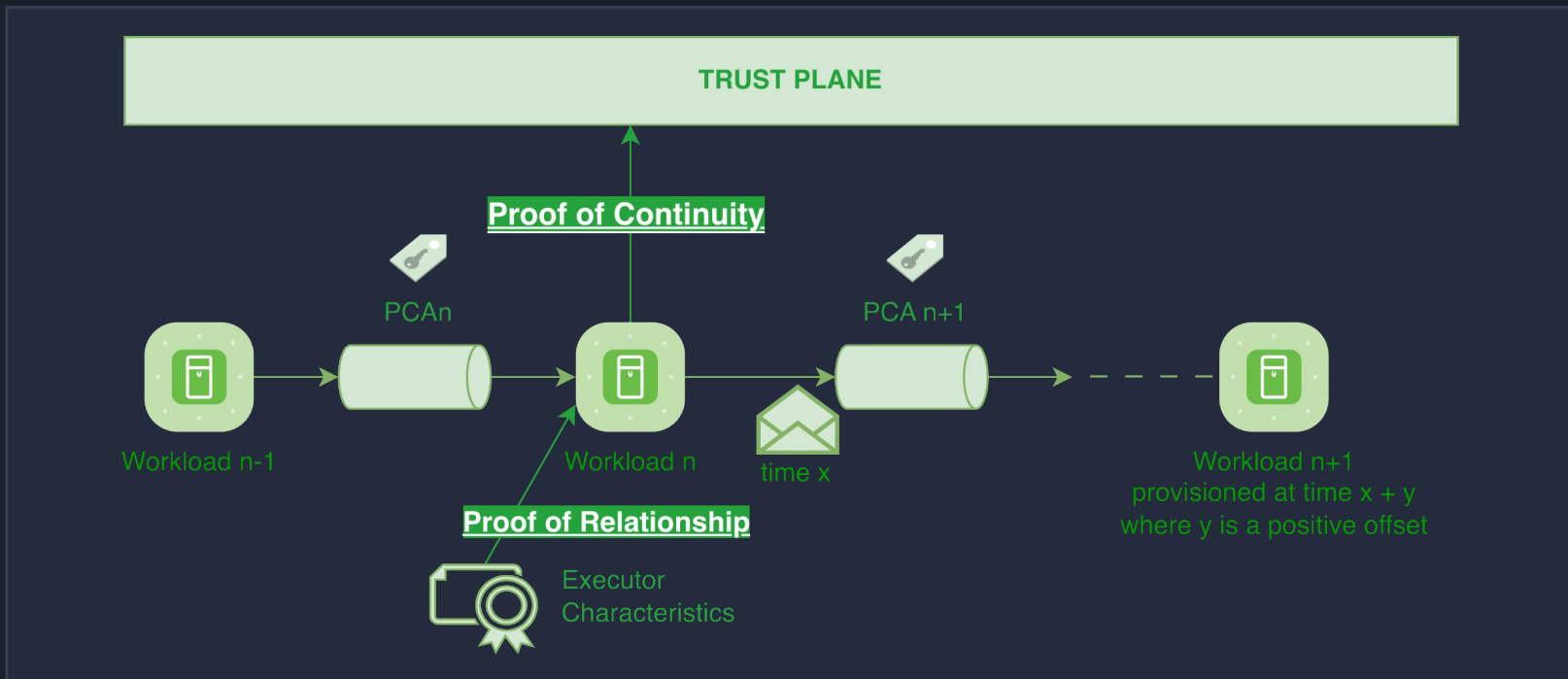
Execution is powered by continuity.

No continuity. No execution.

**This is Authority Continuity.**

# What about the Canonical Execution Model?

*Rethinking the Canonical Execution Model from execution to implementation*



# PIC Specification

The screenshot shows a GitHub repository page for 'pic-spec' in the 'draft/0.1' branch. The file 'pic-spec.md' is selected, and its content is displayed in a preview mode. The page includes navigation links for Platform, Solutions, Resources, Open Source, Enterprise, and Pricing. The repository is public and has 2 forks and 16 stars. The file's metadata is as follows:

- Version:** 0.1 (Draft)
- Status:** Draft – Not a Standard
- Date:** 2025-12-11
- Publisher / Steward:** Nitro Agility S.r.l.
- Model Origin (Theoretical):** PIC Model (Nicola Gallo)
- Source:** [github.com/pic-protocol/pic-spec/draft/0.1/pic-spec.md](https://github.com/pic-protocol/pic-spec/draft/0.1/pic-spec.md)




## At This Point, You Might Ask

Essentially we're saying **Proof of Possession** alone **can't solve everything**, and we're introducing **new primitives**.


... **Does this replace everything?**

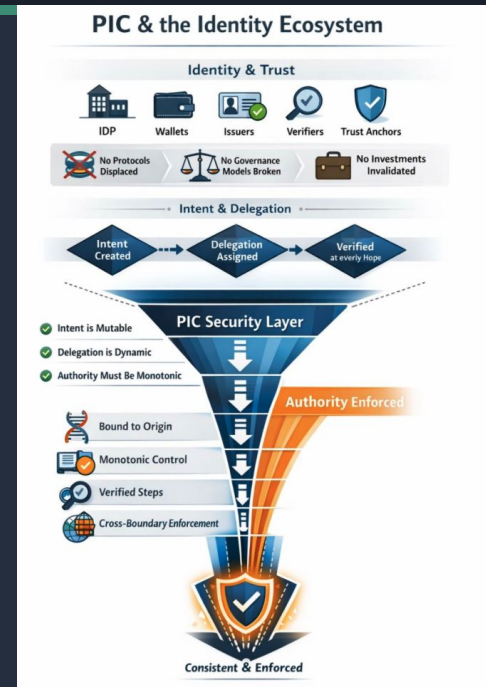
# Provenance Identity Continuity (PIC) Model

## Keep in mind that

 PIC is not a new identity protocol → No changes to the identity infrastructure: IDPs and wallets stay exactly where they are.

 PIC does not interfere with the authentication process.

 It encodes execution into the security model itself → enables authority propagation through execution.



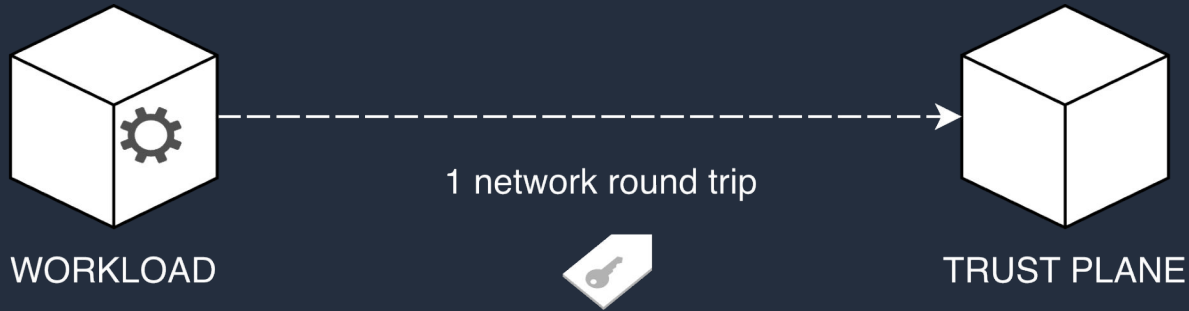


## At This Point, You Might Ask

Is this **acceptable** in **terms** of **latency**?

# PIC vs OAuth Token Exchange: Network Latency

*Latency is not the problem, it's a single network round trip, like the one used for the Authorization Server (IdP)*





## At This Point, You Might Ask

What does this **mean for execution time?**

# Execution Times

```
pic-prototyping — nicolagallo@Host-003 — ..c-prototyping — -zsh — 180x42

Has private key: true
→ Created PCA_0 (396 bytes)
→ Forwarding to Archive
SOVEREIGN-ARCHIVE
DID: did:web:archive.sovereign.example
Issuer: did:web:trustplane.sovereign.example
Role: Executor
Has public key: true
Has private key: true
→ Received PCA hop=0 ops=["read:/user/**", "write:/user/**"]
→ Created PoC (5773 bytes)
→ Received new PCA (685 bytes)
→ Forwarding to Storage
SOVEREIGN-STORAGE
DID: did:web:storage.sovereign.example
Issuer: did:web:trustplane.sovereign.example
Role: Executor
Has public key: true
Has private key: true
→ Received PCA hop=1 ops=["read:/user/**", "write:/user/**"]
→ Created PoC (6062 bytes) - final hop
✓ Written: /user/output_1772290469716.txt
→ Received: /user/output_1772290469716.txt
→ Received: /user/output_1772290469716.txt

20 chain(s) sequential


Chains executed: 20
Hops per chain: 2
Total hops: 40

Total time: 4.74 ms
Per chain: 237.18 µs
Per hop: 118.59 µs

Avg PCA size: 396 bytes
Avg PoC size: 11835 bytes
Avg total/hop: 12231 bytes

~/source/nitro/wg-workspaces/repos/pic-prototyping main > |
```

53s 15:54:29

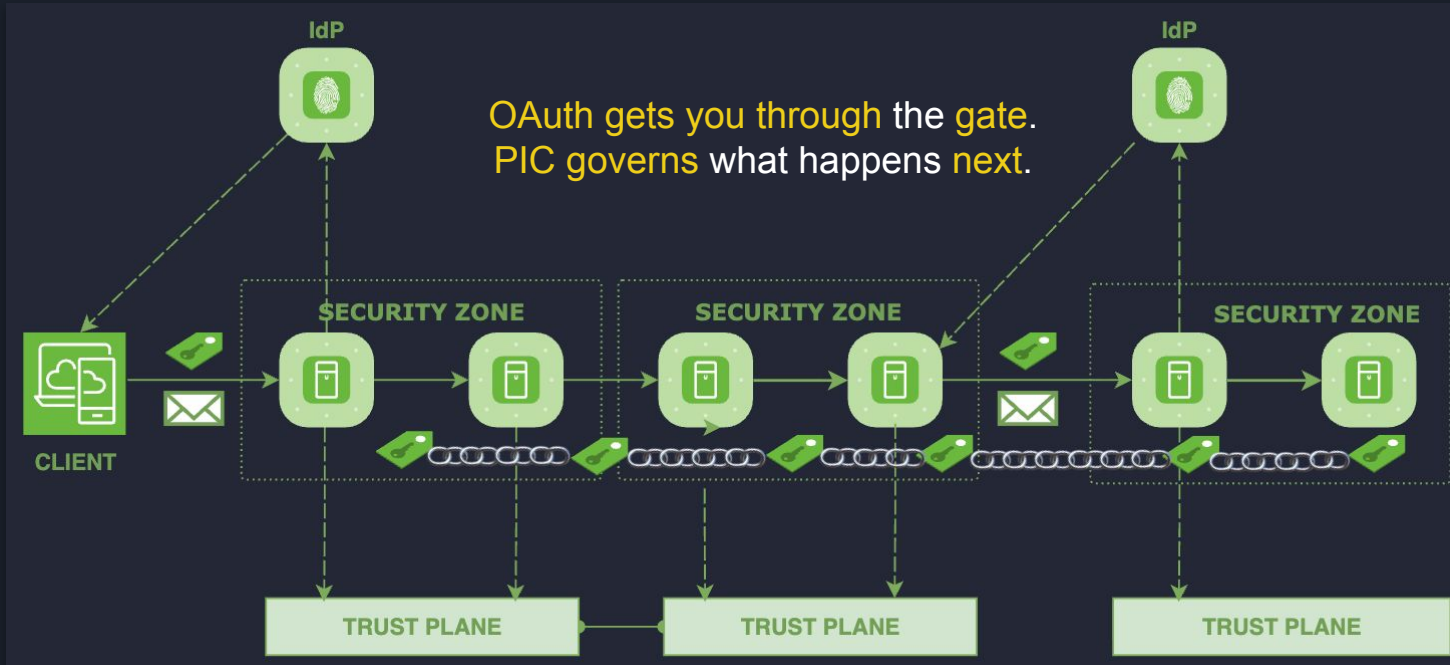


# At This Point, You Might Ask

How would you industrialize this?

# OAuth as a PIC Federation Backbone

*OAuth can serve as the federation backbone, with the PCA encoded into the access token as a custom claim. Ensuring full compatibility with existing infrastructure and applications.*



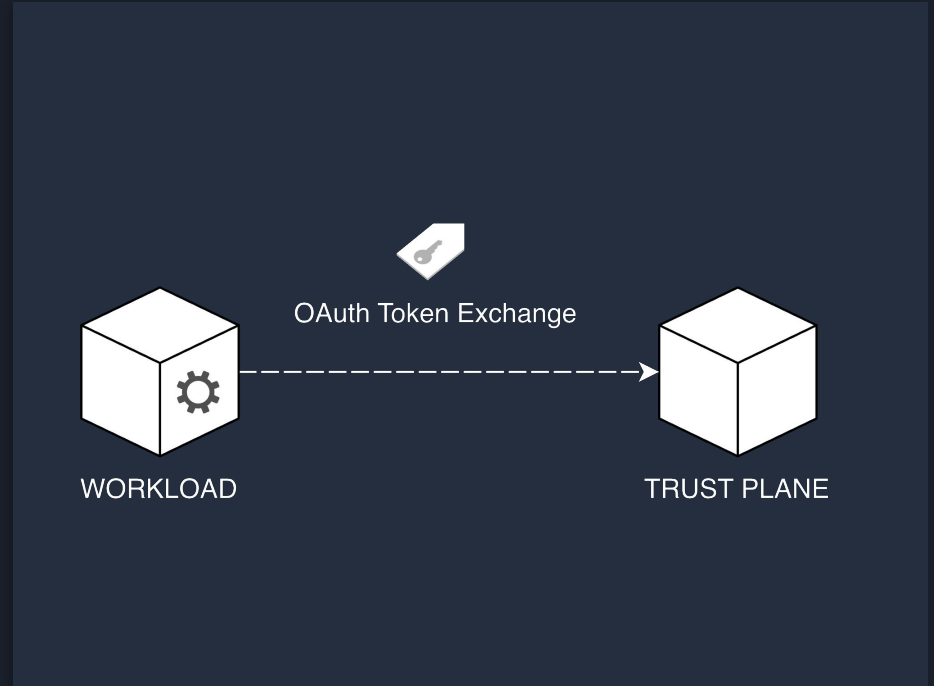


## At This Point, You Might Ask

Are you saying that a **Trust Plane** could also **adopt** an interface like **OAuth Token Exchange** with a **custom profile**, enabling the industrialization path?

# At This Point, You Might Ask

Yes, an **OAuth Token Exchange custom profile** is a viable solution for industrialization.



# PIC DEMO

The screenshot shows the GitHub repository page for `pic-protocol/pic-prototyping`. The repository is public and has 1 branch and 0 tags. The commit history is as follows:

Commit	Author	Message	Time
<code>774ef19</code>	ngallo	updated email	2 months ago
<code>774ef19</code>	ngallo	improved the logic to create the fixtures	3 months ago
<code>774ef19</code>	ngallo	updated deps	3 months ago
<code>774ef19</code>	ngallo	Add Rust prototyping; workload identity g...	3 months ago
<code>774ef19</code>	ngallo	updated email	2 months ago
<code>774ef19</code>	ngallo	working on legal matters	3 months ago
<code>774ef19</code>	ngallo	working on legal matters	3 months ago

The screenshot shows a code editor displaying the `did.json` file. The file content is as follows:

```
1 {
2   "@context": [
3     "https://www.w3.org/ns/did/v1",
4     "https://w3id.org/security/suites/ed25519-2020/v1"
5   ],
6   "assertionMethod": [
7     {
8       "did": "did:web:archive.sovereign.example#key-202512"
9     }
10  ],
11  "authentication": [
12    {
13      "did": "did:web:archive.sovereign.example#key-202512"
14    }
15  ],
16  "id": "did:web:archive.sovereign.example",
17  "verificationMethod": [
18    {
19      "controller": "did:web:archive.sovereign.example",
20      "id": "did:web:archive.sovereign.example#key-202512",
21      "publicKeyJwk": {
22        "crv": "Ed25519",
23        "kid": "did:web:archive.sovereign.example#key-202512",
24        "kty": "OKP",
25        "x": "3CuauxdQdzxzE6Gj72TPuig8TRVh3t6GYwJK8YfACE"
26      }
27    },
28    {
29      "type": "Ed25519VerificationKey2020"
30    }
31  ]
32 }
```

# PIC DEMO

The screenshot shows a code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders like 'wg-workspaces', 'trusted-ai-agents', 'pic-github', 'pic-model', 'pic-protocol.github.io', 'pic-prototyping', and 'fixtures/workload-credentials-test-k...'. The 'did.json' file is selected and open in the editor. The code in the editor is a JSON object representing a DID document. The spell checker interface is visible at the bottom, showing 'No issues found.' for both the main editor and a separate 'SPELL CHECKER ISSUES: WORDS WITH ISS...' panel.

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 11
12 12
13 13
14 14
15 15
16 16
17 17
18 18
19 19
20 20
21 21
22 22
23 23
24 24
25 25
26 26
27 27
28 28
29 29
30 30
31 31
32 32
```

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "assertionMethod": [
    "did:web:trustplane.sovereign.example#issuer-key-202512",
    "did:web:trustplane.sovereign.example#cat-key-202512"
  ],
  "authentication": [
    "did:web:trustplane.sovereign.example#issuer-key-202512",
    "did:web:trustplane.sovereign.example#cat-key-202512"
  ],
  "id": "did:web:trustplane.sovereign.example",
  "verificationMethod": [
    {
      "controller": "did:web:trustplane.sovereign.example",
      "id": "did:web:trustplane.sovereign.example#issuer-key-202512",
      "publicKeyJwk": {
        "crv": "Ed25519",
        "kid": "did:web:trustplane.sovereign.example#issuer-key-202512",
        "kty": "OKP",
        "x": "hvyRnfyB9_BLBLa20npk6uzD0XbNtI2NMqJED__LE"
      },
      "type": "Ed25519VerificationKey2020"
    },
    {
      "controller": "did:web:trustplane.sovereign.example",
      "id": "did:web:trustplane.sovereign.example#cat-key-202512",
      "publicKeyJwk": {
        "crv": "Ed25519",
        "kid": "did:web:trustplane.sovereign.example#cat-key-202512",
        "kty": "OKP",
        "x": "hvyRnfyB9_BLBLa20npk6uzD0XbNtI2NMqJED__LE"
      },
      "type": "Ed25519VerificationKey2020"
    }
  ]
}
```

OUTPUT PORTS BRUNO LOGS GITLENS PROBLEMS 6 DEBUG CONSOLE SPELL CHECKER ISSUES BY FILE TERMINAL

SPELLING ISSUES

No issues found.

SPELL CHECKER ISSUES: WORDS WITH ISS...

No issues found.

Ln 1, Col 1 Spaces: 2 UTF-8 LF { } Json v0.16.1 Autocomplete (0) Prettier

# PIC DEMO

```
1 {
2   "econtext": [
3     "https://www.w3.org/ns/did/v1",
4     "https://w3id.org/security/suites/ed25519-2020/v1"
5   ],
6   "assertionMethod": [
7     "did:web:gateway.sovereign.example#key-202512"
8   ],
9   "authentication": [
10    "did:web:gateway.sovereign.example#key-202512"
11  ],
12  "id": "did:web:gateway.sovereign.example",
13  "verificationMethod": [
14    {
15      "controller": "did:web:gateway.sovereign.example",
16      "id": "did:web:gateway.sovereign.example#key-202512",
17      "publicKeyJwk": {
18        "crv": "Ed25519",
19        "kid": "did:web:gateway.sovereign.example#key-202512",
20        "kty": "OKP",
21        "x": "FC3rpSAjxt73fKS4VLVH_oIXLNG6KkFo_8su4o87MAA"
22      },
23      "type": "Ed25519VerificationKey2020"
24    }
25  ]
26 }
```

OUTPUT PORTS BRUNO LOGS GITLENS PROBLEMS 6 DEBUG CONSOLE SPELL CHECKER ISSUES BY FILE TERMINAL

SPELLING ISSUES

No issues found.

SPELL CHECKER ISSUES: WORDS WITH ISS...

No issues found.


Ln 1, Col 1 Spaces: 2 UTF-8 LF {} Json v0.16.1 Autocomplete (0) Prettier



# PIC DEMO

```
WG (WORKSPACE)
├── wg-workspaces
├── trusted-ai-agents
├── dif-notes
├── pic-github
├── pic-model
├── my-private-papers
├── pic-protocol.github.io
├── pic-prototyping
│   ├── fixtures/workload-credentials-test-k...
│   ├── rust-prototyping
│   │   ├── target
│   │   ├── workload-identity-gen
│   │   └── workload-runner
│   │       ├── benches
│   │       └── src
│   │           ├── workload
│   │           │   ├── sovereign
│   │           │   │   ├── archive.rs
│   │           │   │   ├── gateway.rs
│   │           │   │   ├── mod.rs
│   │           │   │   ├── registry.rs
│   │           │   │   ├── storage.rs
│   │           │   │   └── trustplane.rs
│   │           │   ├── instrumentation.rs
│   │           │   ├── mod.rs
│   │           │   ├── lib.rs
│   │           │   ├── main.rs
│   │           │   ├── Cargo.toml
│   │           │   ├── Cargo.lock
│   │           │   ├── Cargo.toml
│   │           │   ├── gitignore
│   │           │   ├── CODE_OF_CONDUCT.md
│   │           │   ├── CONTRIBUTING.md
│   │           │   ├── GOVERNANCE.md
│   │           │   ├── LICENSE
│   │           │   ├── MAINTAINERS.md
│   │           │   ├── README.md
│   │           │   ├── SECURITY.md
│   │           │   └── Taskfile.yml
│   │           ├── pic-spec
│   │           └── pic-rust
├── ...
└── ...
```

```
trustplane.rs
1  /* Nicola Gallo, 3 months ago | 1 author (Nicola Gallo)
2  * Copyright Nitro Agility S.r.l.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * https://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * Limitations under the License.
15 */
16
17 /// TrustPlane – the Causal Authority for Trust (CAT) in the Sovereign federation.
18
19 use anyhow::{anyhow, Result};
20 use ed25519_dalek::SigningKey;
21 use pic::pca::{
22     CatProvenance, Constraints, CoseSigned, Executor, ExecutorBinding, ExecutorProvenance,
23     PcaPayload, Provenance, SignedPca, SignedPoc, TemporalConstraints,
24 };
25
26 use crate::workload::sovereign::registry::Registry;
27
28 use super::WorkloadIdentity;
29
30 Nicola Gallo, 3 months ago | 1 author (Nicola Gallo) | 1 implementation
31 pub struct TrustPlane {
32     identity: WorkloadIdentity,
33     signing_key: SigningKey,
34 }
35
36 impl TrustPlane {
37     pub fn new(identity: WorkloadIdentity) -> Result<Self> {
38         let signing_key: SigningKey = Identity::new_workload_identity(
39             identity.private_key.clone().ok_or_else(|| anyhow!("TrustPlane requires a private key"))?);
40         Ok(Self {
41             identity,
42             signing_key,
43         })
44     }
45
46     pub fn create_pca_0(
47         &self,
48         ...
49     )
```



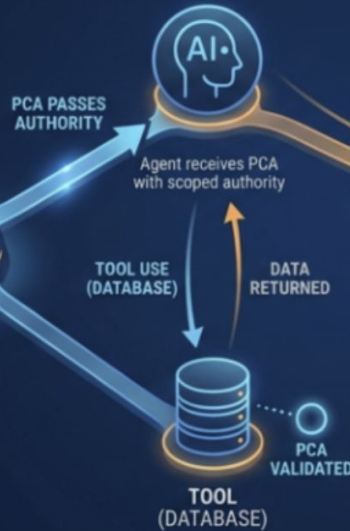
This implementation uses a decentralized approach, but PIC is deployment-agnostic. A centralized implementation based on WIMSE / Attestation is equally valid. We chose the decentralized path only to accelerate the first prototype.

# PROVENANCE IDENTITY CONTINUITY (PIC) PROTOCOL FLOW DIAGRAM - PIC-PROTOTYPING APPLICATION

## 1. USER INITIATES (ALICE)

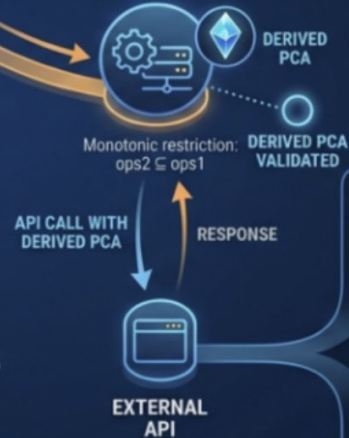


## 2. PRINCIPAL 1 (AI AGENT)



## 3. CAUSAL DERIVATION (HOP 2)

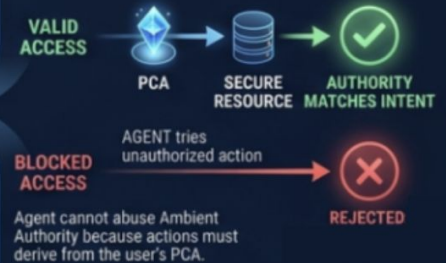
### PRINCIPAL 2 (MICROSERVICE)



## AUTHORITY PROPAGATION (MONOTONICITY RULE)



## ENFORCEMENT (PREVENTING CONFUSED DEPUTY)



Authority is not identity.

Authority is not possession.

**Authority is continuity.**

---

PIC is not just a protocol.

It is a correction of the model.

It is a new security primitive.

# Get Involved



## Website

[www.pic-protocol.org](http://www.pic-protocol.org)



## Specification

[github.com/pic-protocol/pic-spec](https://github.com/pic-protocol/pic-spec)



## Formal Model

Zenodo (DOI: [10.5281/zenodo.17833000](https://doi.org/10.5281/zenodo.17833000))



PIC

[www.pic-protocol.org](http://www.pic-protocol.org)

Not a Policy.  
Not a Token.  
An *Ontological Shift*  
Proven Mathematically.

Thank you!



JOIN THE PIC SLACK CHANNEL

# Appendixes

# Crossing the OAuth Perimeter

Once past the gate, full PIC with Trust Planes. Authority is created, continued, and verified natively. No tokens, no re-materialization, no confused deputy in the security model itself.

## PIC Enables

- ✓ Federated Trust Planes
- ✓ Partitioned security domains
- ✓ Zero-trust by construction
- ✓ Refactoring-invariant security

## No need to rewrite everything

- ✓ Integrates with OAuth / OIDC
- ✓ Works with SPIFFE / DID
- ✓ Federate with existing IdPs
- ✓ Incremental adoption path

## Transitional Path

1. Federate IdP  
with Trust Plane



2. Exchange intent  
for PCA



3. Embed PCA as  
custom claim



4. Enter perimeter  
→ PIC-native