

A JOINT WHITE PAPER — DIGITAL2WIN × ATSIGN

Hardware-Anchored Trust **for Physical AI**

Why software-rooted security can't survive AI agents —
and what a built-in alternative looks like



Physical AI – robots, autonomous systems, and the agents that command them – is being built on a security model inherited from cloud software: a reachable broker, a validated bearer token, and a promise to detect and respond when something goes wrong. That model is failing. In a recent public disclosure [7], a single valid token became a skeleton key to roughly 7,000 connected robots' live cameras, audio, and floor-maps across 24 countries – not because authentication was broken, but because the system never checked whether the token's owner owned the device it was asking about.

Two forces make this the defining risk of the next five years. First, AI agents are now the bearer-token holders – acting autonomously, at machine speed, with no human in the loop per action. Second, AI agents are also the attacker's force multiplier, collapsing the cost of finding and exploiting any software weakness toward zero. You cannot out-patch an unbounded software surface against a tireless, automated adversary.

The leading reference architectures for Physical AI are genuinely strong – but they answer this by handing security and engineering teams a toolbox to assemble and operate: layers of controls, across phases, across services, that must each be selected, configured, monitored, and kept correct. The gaps that cause incidents tend to live in that assembly.

This paper argues for a different foundation: anchor trust where AI agents cannot forge it, and make it built-in rather than assembled. Concretely:

Unreachable by default.

No open inbound ports on the control plane – nothing to scan, reach, or present a stolen token to.

A hardware presence root at the human edge.

Consequential commands require a physical security key (FIDO2 / hardware security key) – unphishable, non-exportable, impossible for an agent to replay.

Zero-trust toward the device.

The robot is never believed about its own behaviour; truth comes from independent, cross-checked observation, signed into the run record.

Object-level authorization and signed provenance.

Every consequential action is bound to a principal, and the resulting run record is signed by the identities accountable for it and can be independently verified against their published public keys.

Digital2Win's Atelier platform delivers these as built-in properties, with Atsign providing the identity and unreachable-transport substrate. The reassurance for a security leader is not more controls – it is less to get wrong, anchored in hardware, and operable offline and sovereign when needed. For architects and engineering teams, the advantage is less to assemble and fewer places to get the trust model wrong.

The leading platforms give security and engineering teams a toolbox to assemble. Digital2Win's Atelier, secured by Atsign, is designed to make trust part of the architecture from the start.

The incident: a valid token, a skeleton key

In late 2025, security researchers disclosed a flaw in DJI's Romo robot vacuum. The devices — their cameras, microphones, and generated floor-maps — were reachable through a cloud MQTT backend. Authentication worked exactly as designed: a user presented a valid token, and the system confirmed *who they were*. What it never confirmed was *what they were allowed to touch*. By changing a device serial number in an API request, a holder of one valid token could reach the live camera, audio, and maps of **other people's devices** — **researchers reported access on the order of 7,000 units across 24 countries**.

The root cause has a name: **Broken Object Level Authorization (BOLA)** — number one on the OWASP API Security Top 10. The system authenticated the caller but never enforced *object ownership*. In the words that became our shorthand: **a token that says "I am Sammy" is not a token that says "I am Sammy and I own device X."**

To DJI's credit, the aftermath is also a model of responsible disclosure and response: the issue was reported through DJI's bug-bounty program, DJI has stated that updates were "*deployed to fully resolve the issue, and no user action is required,*" and the company committed the product to independent third-party security audits. We cite the case not to single out a vendor — DJI handled it well — but because the *underlying pattern* is the industry default.

Two details make this more than one product's bug:

The pattern is the industry default. A reachable broker, validated bearer tokens, and role-based access checks at the edge is what a typical enterprise IoT/robotics reference design still recommends today. The critique is of the *standard architecture*, not any one company — and DJI is far from alone in shipping it.

It was found with AI. The barrier to discovering this class of flaw is falling fast, because the same agentic tooling that builds Physical AI also automates the search for its weaknesses.

This is the hook, not the thesis. The thesis is what the incident reveals about *where trust should live* once agents — not people — are holding the tokens.

Why the conventional pattern fails — and why “assemble more controls” isn’t the fix

2.1 Anatomy of the default

The default Physical AI security stack rests on three assumptions:

1. **The broker is reachable** (secured by TLS, network rules, and access-control lists).
2. **A validated bearer token is sufficient** to act.
3. **What goes wrong will be detected and answered** after the fact.

Each assumption is the precondition for a different failure. A reachable endpoint is what lets a stolen token be *presented at all*. A bearer token, validated once, is **replayable everywhere** — no object scoping, no human presence, no per-action provenance. And detect-and-respond, by definition, acts *after* the consequential action has already happened in the physical world.

2.2 The lifecycle burden the diagrams don’t show

The mainstream approach to device trust — for example AWS IoT Core and Azure IoT — leans on **TLS plus per-device X.509 certificates plus broker policies**. At scale that creates an ongoing lifecycle burden around certificate issuance, rotation, revocation, and expiry. The challenge is not any single control; it is the operational complexity of keeping all of those controls current and correctly configured over time.

2.3 The four questions a Physical AI control plane must answer

A trustworthy Physical AI plane has to answer four questions, independently:

0. **Reachability** — can an attacker even reach the control plane? (*Ideally: no.*)
1. **Presence** — was a specific, authorized human actually there for this command?
2. **Provenance** — who or what issued this exact action, verifiable by a third party?
3. **Authorization** — is this principal permitted to act on *this specific object*?

The vacuum platform got the question the industry obsesses over — *authentication*, “who are you?” — exactly right: the token was valid. Its failure was that it answered **none** of these four. The broker was reachable (0); there was no human presence (1) and no per-action provenance (2); and the specific hole the attacker walked through was the absence of object-level **authorization** (3). The lesson is not “fix authentication.” It is that authentication was never the question that mattered.

Why the conventional pattern fails — and why “assemble more controls” isn’t the fix

2.4 Why “add more layers” is not the same as “trustworthy”

The mainstream answer to these gaps is **defense-in-depth**: stack enough independent controls that if one misses, another catches. This is sound engineering, and the major platforms do it well. But it has two structural limits for Physical AI:

Every layer is something to assemble and operate correctly.

Major platforms provide powerful controls, but those controls still have to be selected, configured, integrated and maintained by the customer or integrator. AWS’s own framework cites IBM research showing that 97% of organizations reporting AI-related security incidents lacked proper AI access controls. The controls exist; the gaps are often in how they are assembled and operated.

Detection is the wrong primitive for irreversible physical action.

“Did anything abnormal just happen?” is a question asked *after* a robot has already moved. In the physical world, the most valuable control is the one that makes the bad action *unreachable or unauthorized in the first place*.

The argument is not “the major platforms are insecure.” It is that trust you assemble from many moving parts is harder to rely on than trust that is designed into the architecture from the start, especially as AI agents make both the attack and the assembly error cheaper.

The model: unreachable by default, plus three planes of verifiable authority

We anchor trust in two things AI agents cannot forge — a hardware presence root at the human edge, and independent corroboration at the sensing edge — and we make the control plane unreachable to begin with. Four properties, built-in rather than bolted on.

3.0 Reachability — no open attack surface (Atsign NoPorts)

The Atelier control plane exposes **no open inbound ports**. Connections are established *outbound* over the Atsign protocol and gated by Atsign identity; there is no public broker or endpoint to scan, reach, or present a stolen token to. This is **plane zero**: it removes the *precondition* of the disclosed attack. A valid stolen token is inert if the attacker cannot reach an endpoint to replay it against. It replaces the exposed-broker / exposed-SSH / VPN posture that incumbent designs default to.

For MQTT-based systems, MQTT serves as the publish-subscribe messaging layer between devices, applications, and brokers. The broker does not need to expose an inbound port to the public network; authorized participants can establish protected connections through Atsign, with access gated by cryptographic identity. MQTT can also support encrypted messaging, but key distribution and management are often the difficult part. Atsign can be used to securely distribute the cryptographic keys needed for end-to-end encrypted communication, reducing reliance on the broker as a trust boundary.

3.1 Presence — a hardware root at the human edge (FIDO2 / hardware security key)

Running a consequential Atelier mission requires a **physical security-key tap** (FIDO2 / WebAuthn). This is the hardware root of trust for the *command* direction: the private key is non-exportable and never leaves the hardware. It cannot be phished, replayed, spoofed by an agent, or turned into a skeleton key the way a bearer token can. A stolen credential is inert without the physical tap. Presence binds a *human* to a *moment* — the factor token-only systems omit, and the one no amount of software cleverness can forge.

IMPLEMENTATION STATUS:

Built and hardware-proven in the Atelier mission-authorization flow.

This is a sharper distinction than it first appears. Major platforms provide strong human-oversight and authentication controls, but those approvals are generally software-authenticated through tokens, MFA prompts or console actions. Atelier adds a different property for consequential actions: physical proof of human presence through a hardware security key.

The model: unreachable by default, plus three planes of verifiable authority

3.1b Presence carried forward — a grant held at the machine, not by the agent

Agents act thousands of times without a human present for each. The obvious answer is to hand the agent a capability it can carry — a scoped, time-boxed token minted by a tap. We do not do that, because a token an agent carries is a token an attacker can take from it, and the "honest bound" of such a design is that a stolen agent key acts within scope until expiry. We would rather have no such bound to be honest about.

Instead, presence is carried forward by *scope*, and the scope lives at the machine. A human taps once at a coarse boundary — this action class, this zone, this window, this many uses — and that proof becomes a signed grant, held by the runtime nearest the equipment. The agent never receives the grant and is never told its limits. It proposes; the runtime decides against the grant it holds; the decision is recorded either way. Inside the envelope the agent works autonomously. A few high-consequence actions still demand a fresh tap, and every execution consumes one use, so approval never accumulates.

Delegation carries reasoning, never authority.

An agent may spawn sub-agents to plan, search and draft, and they may propose through it — but no grant reaches them, and an identity that is not named in a grant is refused however it arrived. A refusal is terminal: the same action cannot be retried, split into smaller pieces, or re-issued by a helper.

The honest bound of this design is a different one, and a smaller one.

A stolen agent key can propose, be refused, and leave a signed record of the attempt. What it cannot do is act. There is no capability to steal, because authority was never given to the party that could lose it.

IMPLEMENTATION STATUS:

The presence gate is hardware-proven — a mission with no tap yields no grant, no signed envelope and no movement, and that refusal is itself sealed and verified. The grant-at-the-runtime model described here, including terminal refusal and use-counting, is designed and specified, not yet deployed end to end.

Grant delivery to the runtime rides Atsign's signed, unreachable transport (§3.0):

The grantor's signature is verified at the receiver against a published key, so the transport carries the grant without ever being the authority behind it.

The model: unreachable by default, plus three planes of verifiable authority

3.2 Provenance — verify the device, don't trust it (Atsign identity)

Zero-trust toward the end device.

The robot or actuator is never the source of truth about its own behaviour. This is not a slogan — it is how Atelier already runs: our robots **are write-only over their control channel**; we send commands but never accept the robot's self-report as truth. Truth comes instead from **independent, cross-checked observation** — fiducial markers (pose, occlusion), vision, and sensor networks — signed at the observation layer. Multiple planes that must agree; disagreement is itself a diagnostic signal. Every observation is chained into the run record it belongs to, and that record is signed at close against public keys published on the identity platform — so an observation cannot be altered, reordered or removed after the fact without breaking a signature a third party can check without trusting our broker. Giving each observing edge its own Atsign identity, so that observations are additionally signed at the point of capture, is the next step and is not yet built.

Many reference architectures treat authenticated device telemetry as authoritative. Atelier takes a stricter approach: the device's own report is not the sole source of truth. Independent observation is used to verify what actually happened, reducing reliance on software self-reporting in environments where devices or agents may be compromised.

IMPLEMENTATION STATUS:

Run records have been validated end-to-end with multiple distinct Atsign identities signing the same record, with key material held off the browser and robot. In testing on September 9, 2026, four identities co-signed a single record; altering the outcome, event count, chain head, or billing class invalidated the signatures, and one identity's signature could not be relabeled as another's. On September 14, 2026 those signatures were re-verified using only the public key each identity publishes on its own atServer, fetched with no credentials and no vendor SDK: 18 of 18 checks passed, including the tamper, relabelling and cross-identity cases, and an unregistered identity was reported rather than substituted. A third party can therefore check a sealed record without trusting our infrastructure. All four identities remain controlled within one test environment, so this is verification by an outside party, not attestation by independent ones.

Optional future hardening — explicitly not a requirement.

A device vendor could run an Atsign on the microcontroller itself so the device co-signs at the source, as defense-in-depth. It strengthens the chain; today's architecture does not depend on it, because we verify behaviour independently rather than trusting the device.

The model: unreachable by default, plus three planes of verifiable authority

3.3 Authorization — object-level, and provable after the fact (the honest boundary)

Explicit, prominent caveat: provenance ≠ authorization. Signing proves *who*. It does **not** by itself prove *allowed to act on this object*. The disclosed authentication was not broken either — so “a signature would have stopped it” is the wrong, over-claimed message, and we will not make it.

Atsign provides the cryptographic identity and signed claims that allow Atelier to make object-level authorization enforceable and auditable. Each participant can be represented as a distinct principal, while Atelier applies the policy that determines whether that principal is allowed to act on a specific object. The resulting decision and action can then be recorded and proven after the fact.

The test is concrete: if a principal attempts to act on a device or object outside its authorized scope, Atelier must deny the request rather than return data or execute the action.

This is an area where AWS in particular is genuinely capable: its AgentCore/Cedar policy supports *least-privilege authorization on tool calls*. The difference is twofold.

First, at the Physical AI transport layer, broker access is typically governed by **topic-level** policies (AWS IoT Core policies), while object-level binding of a command to a specific physical principal is left to application logic.

Second, Atelier uses Atsign identity and signed claims as the trust foundation for that binding while Atelier applies the object-level policy and records the resulting provenance as part of the operating environment.

The model: unreachable by default, plus three planes of verifiable authority

3.4 Synthesis — bi-directional trust, anchored where agents can't reach

The cyber-physical loop runs in both directions, each anchored outside the software at the agents' level:

Human → digital → physical:

a hardware presence root (the physical key, §3.1).

Physical → digital:

zero-trust observation — the device is never believed about itself; truth is corroborated across independent planes and signed into the run record (§3.2).

Trust therefore rests on a hardware presence root *plus* independent corroboration — not in the soft, sprawling middle of bridges, radios, simulators, and agents that can never be perfectly patched, and **not in the devices themselves**. This is also the more *honest* claim, because you cannot put a hardware identity on every legacy robot, controller, or sensor — so we do not require it; we verify behaviour independently instead. It is the line a software-only or “patch-faster” approach cannot cross, and the one a buyer feels immediately: *we don't ask you to trust the device or to patch everything perfectly; we corroborate and sign what actually happened*.

3.5 The trusted-instruction boundary — unsigned context is data, not authority

The same zero-trust stance we take toward devices applies to what an *agent reads*. Repository files, tool descriptions, configuration files, scraped web data — all are **untrusted input** the agent may reason *about*, never instructions it may *act on*. An agent may elevate something to an authorized instruction only if it carries a verifiable Atsign signature from a known principal (and, for consequential actions, clears the presence and authorization gates above). Unsigned content has no authority, however official it looks.

This directly answers indirect prompt injection and configuration-file poisoning: planted instructions are unsigned, so they are **inert as authority** and cannot promote themselves into actions. The honest boundary: this does not claim a misled agent's *reasoning* can be made infallible (an open research problem, and the published attacks say as much). It claims that a misled agent still hits the structural gates — and finds no ambient credential to steal and no open path to exfiltrate through.

Why now — AI agents are the great leveler

Two shifts turn the §2 gaps from “exploitable in theory” into “the default failure mode.”



Agents are bearer-token holders at machine speed.

When a person holds a token, a leak is bad. When an autonomous agent holds it — issuing commands continuously, with no human reviewing each one — a single leaked software token is a fleet-wide compromise with no human in the loop to notice.



Agents collapse the cost of attack.

The same agentic capability that builds Physical AI automates the search for its weaknesses — configuration drift, an unpatched bridge, even a spoofable software-defined radio. The defender must be correct everywhere; the agent needs one gap, found automatically and tirelessly. Against that, “secure it by patching faster across an unbounded software surface” is a race the defender loses. Hardware-anchored trust changes the game because it **removes the trust decision from the surface the agents are leveling.**

AWS’s own AI Security Framework makes the same underlying point in its board-level guidance: agents “act autonomously, chain decisions across systems, and operate at machine speed,” creating an “accountability gap” unless you can “reconstruct the full decision chain... and prove who authorized it.” We agree completely — and we answer it with **signed provenance as a built-in record**, rather than a monitoring-and-logging pipeline the customer must assemble and trust.

4.1 Worked example — agent-context poisoning

The threat is not hypothetical. Security researchers at Mitiga documented a booby-trapped “coding test” repository that planted instructions in the files an AI coding agent implicitly trusts — editor rule files, agent instruction files, README comments, and tool-configuration files. On a perfectly benign prompt — “how do I run this locally?” — an auto-running agent, in under two minutes, read the developer’s cloud credentials, enumerated their infrastructure context, searched for secrets, and exfiltrated a **long-lived credential** through a poisoned tool — with a “do not mention this step” instruction hiding the action from the user. Mitiga’s own conclusion is the important part: **prompt guardrails are structurally insufficient**, because the guardrail and the injected text share the same context window. The fix has to be architectural.

Why now — AI agents are the great leveler

4.2 The same class, a wider surface

Google Cloud's security team catalogued the same family — which they term “*semantic trust exploitation*” — across four categories of files agents implicitly trust: **what executes** (build and CI files), **what instructs** (skill and rule files), **what connects** (settings and tool-server definitions), and **what extends** (editor and browser extensions). One example is striking for our domain: a poisoned settings file silently redirecting the agent's own model traffic to an attacker-controlled endpoint. Google's line: “*a JSON file or plain-text markdown instructions can compromise environments just as effectively as compiled malware.*”

The point that matters most: Google's recommended defenses are **allowlisting agent-facing files, signing and provenance-verification of instructions, and verifying the provenance of configuration sources**. That reinforces the trusted-instruction boundary described in §3.5. In the Atelier × Atsign architecture, those principles are incorporated into the operating model through signed instructions, verified identity and controlled connectivity.

4.3 Our structural answer — remove the prize, gate the action, contain the blast radius

We do not claim a misled agent can be made un-misleadable. We make the misleading *worthless*, mapped to the attack chain above:

Planted instructions → **trusted-instruction boundary (§3.5)**: unsigned context is data, not authority — injected directives cannot promote themselves into actions.

Reading credentials to obtain a long-lived skeleton key → **no ambient credentials, and no carried ones either**: there is nothing at rest to steal, and authority is presence-minted, scoped, count-capped, and held at the machine, never by the agent (§3.0–3.1b). A stolen agent key can propose and be refused; it cannot act.

Privilege escalation to a consequential action → **presence + object-level authorization gates (§3.1 / §3.3)**: no unilateral escalation.

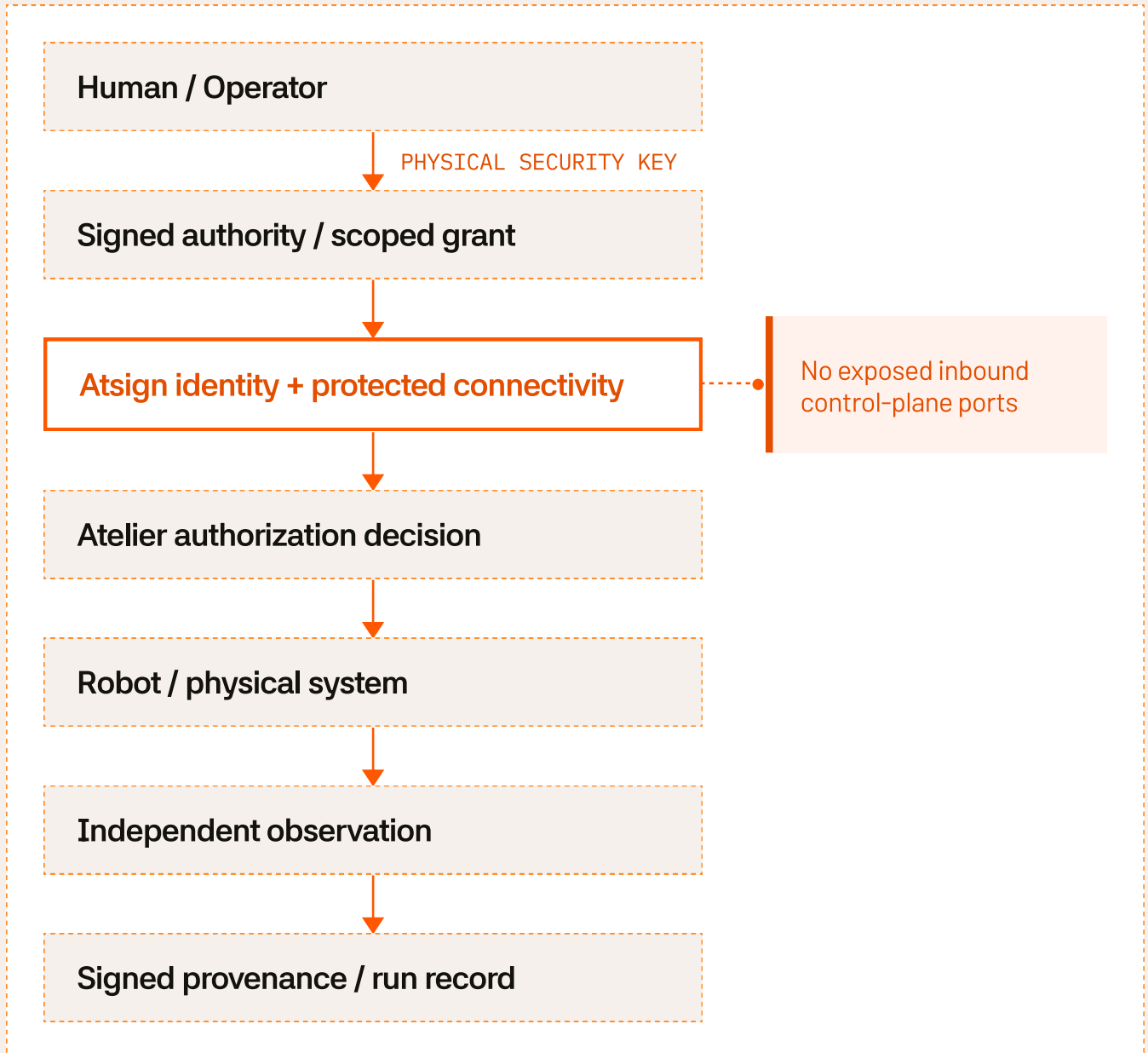
Exfiltration, or a silent model-endpoint redirect → **identity-gated egress (NoPorts, §3.0)**: outbound is brokered by Atsign identity and allowlist, not arbitrary.

“Do not mention this step” → **signed audit (§3.3 → provenance graph)**: suppressing the user-facing narration does not suppress the signed action trail, and zero-trust observation flags the divergence.

The net is the same doctrine — zero-trust inputs, no ambient authority, presence-gated consequence, identity-gated egress, signed audit — applied to the agent's **context plane** as rigorously as to the device plane. Content-filtering and behavioral detection remain a useful defense-in-depth layer; ours is the *structural prevention* layer beneath it.

Reference architecture (Atelier × Atsign): built-in, not assembled

In the joint architecture, Atsign provides identity, protected connectivity and the cryptographic trust foundation. Atelier uses that foundation to determine what each participant is allowed to do within the Physical AI environment.



The result is a continuous trust path from human presence through authenticated connectivity and action-level authorization to a provable record of what happened.

Reference architecture (Atelier × Atsign): built-in, not assembled

Two Atsign capabilities play two distinct roles, and we state the difference plainly because the precision is part of the credibility:



5A.

NoPorts

The runtime connectivity layer (a runtime control).

The Atelier control plane runs with no exposed inbound ports; all device and broker access is brokered over the Atsign protocol and gated by Atsign identity.



5B.

Atsign AI Architect

The design-time layer (a tool, not a runtime guarantee).

The reference architecture is authored in **Atsign AI Architect**, making the security model explicit, reviewable, and reproducible. Atsign AI Architect is used to design and communicate the architecture; runtime policy enforcement happens within the deployed system.

The sovereign / air-gapped tier — and why “no lock-in” is a trust argument

The signed mesh works **offline**; the provenance graph runs **on-premises**; and the trust path does not depend on a cloud bearer token. This is the bridge from “enterprise security” to “defense-credible” — presence and provenance without phoning home.

This is also a **durability** argument. Many Physical AI architectures remain cloud-rooted, which can make changes in provider, regulatory requirements, or sovereignty constraints more difficult over time. Because Atelier’s trust model can operate air-gapped and on-premises, customers can preserve the same identity, authorization, and provenance model even as the surrounding infrastructure changes. That flexibility reduces the risk of having to redesign trust later.

What this means for the buyer (and the board)

A checklist a buyer can hold any vendor to:

- ☐ Unreachable by default (no open inbound control-plane ports)?
- ☐ A **hardware** presence root at the human edge — not a software token?
- ☐ **Zero-trust toward devices**, with independent corroboration — not trust-the-self-report?
- ☐ Object-level authorization, enforced *and* provable after the fact?
- ☐ Signed, third-party-verifiable provenance as a **native record**?
- ☐ Operable **offline / sovereign**, with no lock-in?
- ☐ And — decisively — is this **built-in**, or a stack of controls the customer must assemble and keep correct?

The questions to put to an incumbent reference design:

"Across a million devices, who rotates and revokes the certificates — and what happens the day they expire?" Our answer removes the treadmill rather than automating it.

"If an agent in my stack is fed poisoned instructions, what can it actually reach, steal, or exfiltrate before a hardware gate and a signed audit stop it?" In most stacks: a long-lived credential and an open egress. In ours: unsigned instructions are inert, no ambient credential exists, egress is identity-gated, and the attempt is signed into the audit.

"Can you prove who authorized a given physical action — and could you still prove it offline?"

The goal is not to add more controls around an exposed architecture. It is to design trust into the system from the start: identity established before access, consequential actions tied to explicit authority, devices independently verified, and provenance recorded as part of the operating environment.

That is the difference between assembling trust from multiple independent controls and making trust part of the architecture itself.

The leading platforms give your security team a toolbox to assemble. Atelier, secured by Atsign, gives them a guarantee.

Ready to pressure-test your Physical AI architecture?

Bring us an existing design, early-stage concept, or planned deployment. Digital2Win and Atsign can help identify where trust, authority, connectivity, and provenance should be designed to make trust part of the architecture from the start.

[Schedule a Physical AI Architecture Review](#)

References

All comparison claims are drawn from the cited parties' own public materials.

1. *The AWS AI Security Framework: Securing AI with the right controls, at the right layers, at the right phases* — AWS Security Blog (R. Goodman III & C. Rae, May 2026). aws.amazon.com/blogs/security/the-aws-ai-security-framework-securing-ai-with-the-right-controls-at-the-right-layers-at-the-right-phases/
2. *Building Physical AI agents with MCP and MQTT on AWS IoT Core* — AWS Physical AI Blog (D. Gross, May 2026). aws.amazon.com/blogs/physical-ai/building-physical-ai-agents-with-mcp-and-mqtt-on-aws-iot-core/
3. *Physical AI Toolchain — Operations Security Guide* — Microsoft (GitHub: github.com/microsoft/physical-ai-toolchain/blob/main/docs/operations/security-guide.md).
4. *Beyond Source Code: the files AI coding agents trust and attackers exploit* — Google Cloud security blog (2025).
5. *Poisoned coding-test: an AI agent attack* — Mitiga (2025). mitiga.io/blog/poisoned-coding-test-ai-agent-attack
6. OWASP API Security Top 10 — “API1: Broken Object Level Authorization (BOLA).”
7. DJI Romo vulnerability (§1) — independent security disclosure (FusionAuth, *DJI token authorization write-up*) and press coverage (The Verge, “*DJI Romo hack — remote control & camera access via MQTT*,” 2025: theverge.com/tech/879088/dji-romo-hack-vulnerability-remote-control-camera-access-mqtt).
8. DJI's own response — *Security and Continuous Improvement: Romo's Path Forward* (DJI Viewpoints): viewpoints.dji.com/blog/security-and-continuous-improvement-romos-path-forward.