Agent Permission Protocol (APP)

Formal protocol whitepaper (proposal by Crittora)

Executive summary

Agentic AI systems now initiate actions, invoke tools, and execute workflows across multiple services. This shift breaks the traditional security boundary built around identity and long-lived credentials. The central risk is authority: what an agent is allowed to do, for how long, on whose behalf, and under what constraints.

Without an explicit, execution-time authority layer, agentic AI systems remain fundamentally unsafe regardless of model alignment, prompt constraints, or identity controls.

The Agent Permission Protocol (APP) defines a cryptographically signed and encrypted permission policy plus a deterministic verification and enforcement process that MUST gate agent execution before any action occurs. APP makes authority explicit, time-bound, and verifiable, enabling safe autonomy without relying on model compliance or implicit trust.

Most agent frameworks today are insecure by construction. They grant authority implicitly through tool availability, rely on model compliance for enforcement, and provide no cryptographic proof of authorization. These systems cannot be made safe through better prompts, monitoring, or alignment alone.

Any agent system that allows tools to be invoked without presenting a sealed permission policy at execution time is operating with ambient authority and cannot provide provable safety guarantees.

APP defines the authority layer for agentic systems. Just as TLS secures transport and OAuth standardizes delegated identity, APP standardizes executable authority. Without an explicit authority layer, autonomous agents cannot be made safe at scale.

This paper is a proposal from Crittora to formalize APP as a cross-platform protocol standard. It defines the model, schema, verification order, and conformance expectations required to make agent actions provable, bounded, and auditable.

Abstract

APP is a protocol for explicit, capability-based authority in agentic AI systems. It separates intelligence from authority by requiring a signed and encrypted permission policy before any action-capable execution. Permission policies enumerate allowed capabilities, bind audience and intent, and expire by default. A deterministic validation

pipeline ensures fail-closed enforcement at runtime. This draft presents the protocol model, core semantics, and a path toward standardization.

Authority is not just a security primitive. It is a measurable, auditable unit that enables accountable automation and defensible economics. When authority is explicit, time-bound, and verifiable, risk can be quantified, compliance can be proven, and autonomous execution can be governed with precision.

Conformance language

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as specified in RFC 2119.

Figures are illustrative and non-normative unless stated otherwise.

1. Introduction

1.1 The rise of agentic AI

Agentic systems now execute multi-step workflows and tool calls with minimal human intervention. As autonomy increases, actions and side effects expand beyond what traditional security models can govern.

1.2 Why authority replaces identity as the boundary

Identity can authenticate who is calling a system, but it cannot determine what an agent is allowed to do in a specific context. Authority is the true boundary in agentic execution.

APP defines the authority layer for agentic systems - a layer distinct from identity, transport security, model context, and model reasoning.

This separation of intelligence and authority is enforced outside the model (see Figure 2).

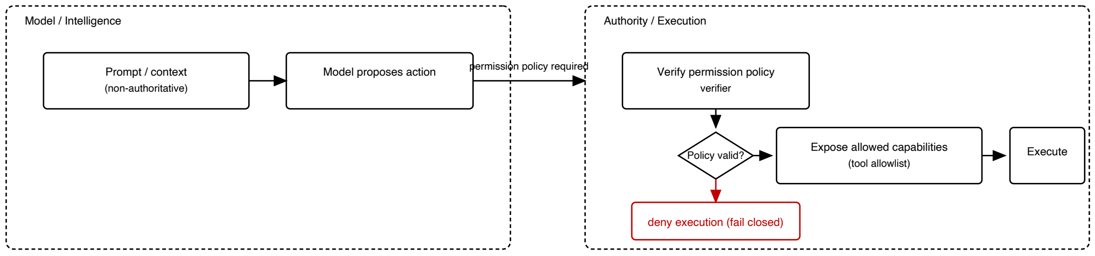

Figure 2. Separation of intelligence and authority. The model may propose actions, but authority is enforced outside the model by pre-execution verification and capability exposure.

1.3 Ambient authority and implicit permissions

Most agent runtimes mount tools by default. Once tools exist in the runtime, authority is implicitly granted without explicit intent or time bounds. This creates ambient permission surfaces that are difficult to audit or constrain.

1.4 A protocol-level solution

Prompt guardrails are advisory and rely on model compliance. APP proposes a protocol boundary: no action occurs without a valid, explicit, verifiable, encrypted permission policy. Any agent system that permits tool invocation or external action without presenting and verifying a sealed permission policy at execution time is operating with ambient authority and cannot provide provable safety, containment, or audit guarantees.

1.5 Proposal scope and goals

APP focuses on explicit authority for agent actions. It does not govern model alignment, tool correctness, or internal reasoning. It is designed to be platform-neutral and enforceable by independent runtimes.

2. Problem statement: authority in agentic AI

2.1 Tool availability equals authority

Agents can invoke any mounted tool, even when that tool is unrelated to the current request or intent.

2.2 Implicit trust assumptions

Agent frameworks frequently assume the model will follow instructions and avoid sensitive actions. This is not a security guarantee.

2.3 Privilege creep

As systems evolve, tools accumulate while permissions rarely shrink. Agents inherit increasingly broad authority over time.

2.4 Replay and context collapse

Prompts and instructions are often reusable and lack binding to time, actor, or intent. This enables unintended reuse and replay.

2.5 Confused deputy in multi-actor systems

Agents act on behalf of multiple users or systems, yet authority is implicit. This leads to confused deputy conditions and unintended delegation.

2.6 Why prompt guardrails are insufficient

Prompt constraints do not prevent tool access at the execution layer and provide no cryptographic assurance or replay protection.

2.7 Capability-based security as the right model

Capability-based models treat authority as an explicit, unforgeable grant. They map cleanly to agent actions and enforce least privilege by default.

3. Design principles

1. Explicit authority over implicit trust.
2. Authority is separate from intelligence.
3. Least privilege by construction.
4. Time-bounded authority by default.
5. Capability-based scope, not role-based access.
6. Deny-by-default execution.
7. Cryptographic verifiability.
8. Replay resistance and single-use authority.
9. Deterministic, auditable enforcement.
10. Protocol-level, not platform-specific.

4. Protocol overview

APP defines a permission policy as the unit of authority. A policy binds intent, scope, audience, and time bounds into a cryptographically verifiable artifact. Policies are presented to an enforcement point that verifies the policy and exposes only the capabilities allowed for the specified duration.

High-level flow:

1. Issue: an issuer constructs a permission policy for a specific intent.
2. Seal: the permission policy is signed and encrypted.
3. Present: the permission policy is transmitted to a runtime for execution.
4. Verify: the runtime validates the permission policy deterministically.
5. Execute: only allowed capabilities are exposed.
6. Audit: verification results and outcomes are recorded.

The end-to-end execution path is shown in Figure 1.

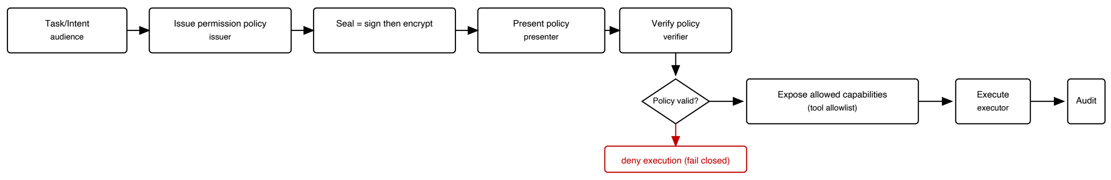

Figure 1. APP execution flow. No agent action or tool invocation is permitted unless a sealed permission policy is presented and verified prior to execution.

Message boundaries and actor interactions are shown in Figure 7.

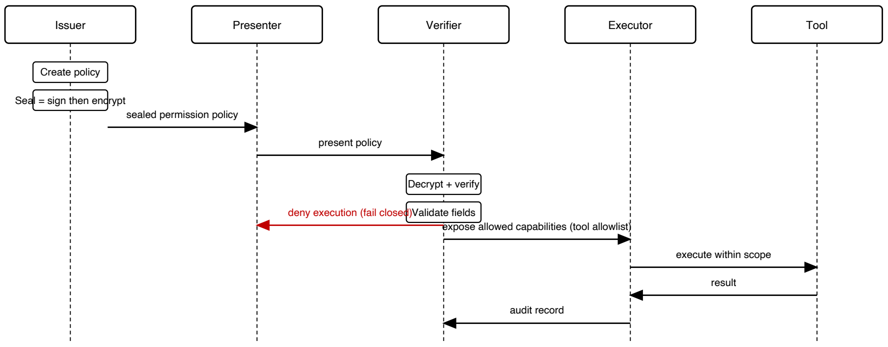

Figure 7. APP sequence diagram. Issuer, presenter, verifier, executor, and tool interactions show policy sealing, presentation, verification, and execution gating.

5. Core concepts

- Capability: explicit authorization to perform a class of actions.
- Permission policy: a structured artifact that encodes authority.
- Intent: the purpose for which authority is granted.
- Scope: the enumerated capabilities permitted.
- Audience: the agents authorized to act.
- Issuer, presenter, verifier, executor: roles in the permission policy lifecycle.

6. Permission policy model

Permission policies are machine-readable documents with required fields:

- type
- policy_version
- request_id
- expires_at
- intent
- audience
- scope

Optional fields include:

- nonce (replay protection)
- limits (runtime constraints)
- metering (optional metadata)

All permission policies MUST be signed and MUST be encrypted. Unencrypted permission policies MUST be denied. Encryption ensures that intent, scope, and authority semantics are not observable by intermediaries or unauthorized parties.

See Figure 3 for the policy structure and sealing requirements.

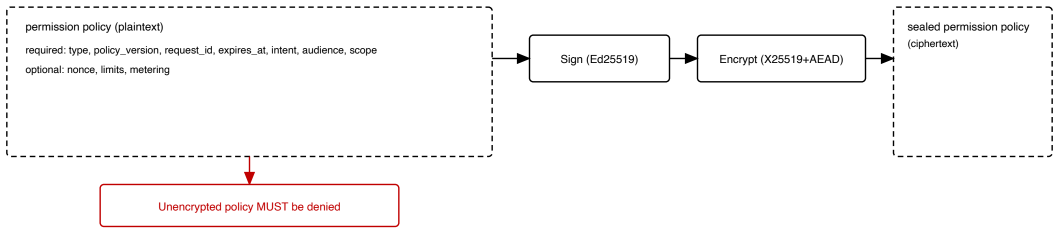
permission policy (plaintext)
required: type, policy_version, request_id, expires_at, intent, audience, scope
optional: nonce, limits, metering
Sign (Ed25519)
Encrypt (X25519+AEAD)
sealed permission policy
(ciphertext)
Unencrypted policy MUST be denied

Figure 3. Permission policy structure. A permission policy encodes intent, audience, scope, and expiration, and is sealed via sign-then-encrypt.

7. Verification and enforcement

Validation is deterministic and fail-closed. A compliant verifier:

1. Decrypts and verifies signature.
2. Parses the permission policy and checks required fields.
3. Validates policy version and expiration.
4. Enforces replay requirements.
5. Enforces audience binding.
6. Exposes only allowed tools and capabilities.
7. Applies runtime limits when present.

Any failure results in denial.

The deterministic validation order is shown in Figure 4.

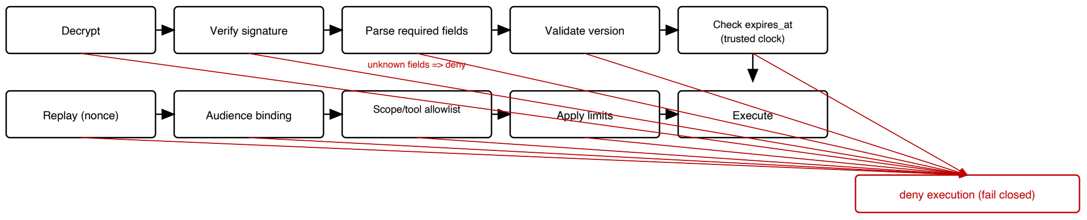

Figure 4. Verifier pipeline (fail closed). Verification is deterministic and denies execution on any cryptographic, semantic, or policy validation failure.

8. Cryptographic profile (APP-Crypto-Profile-1)

To prevent incompatible or insecure implementations, APP defines a mandatory cryptographic baseline for version 1:

- Policy serialization: JSON
- Signing: Ed25519
- Encryption: hybrid encryption with an AEAD payload (X25519 + AEAD)
- Ordering: sign then encrypt (mandatory)

Implementations MAY support additional algorithms, but MUST support this profile for conformance.

9. Threat model and security outcomes

APP is designed to mitigate:

- ambient authority and tool leakage
- replay and unauthorized reuse
- confused deputy scenarios
- unbounded or indefinite permissions
- unverifiable audit trails

Security outcomes include explicit authority, bounded execution, and provable enforcement decisions.

Threats and corresponding APP controls are summarized in Figure 5.

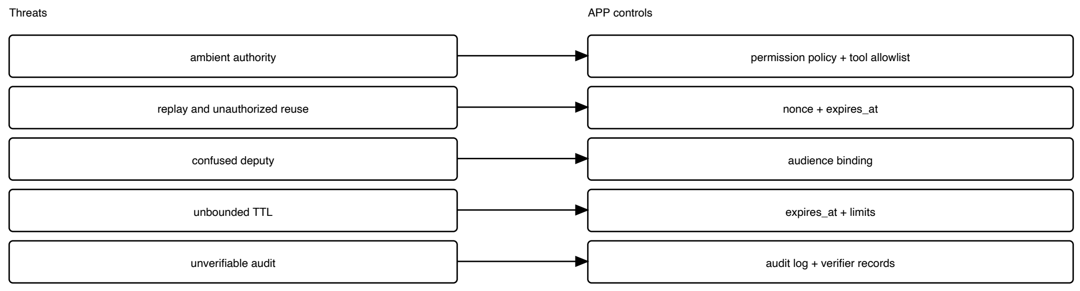

Figure 5. Threat-to-control mapping. APP mitigations map directly to common agentic security failure modes.

10. Integration patterns

APP is compatible with existing tooling stacks:

- Agent runtimes gate tool exposure on permission policies.
- API gateways verify permission policies before forwarding requests.

- Orchestrators enforce permission policies per step in multi-stage workflows.

APP complements OAuth, RBAC, and IAM by providing an explicit authority object for agent actions.

Any system that allows agents to execute actions without presenting a sealed permission policy at execution time is operating with ambient authority, regardless of how its permissions are configured.

Figure 6 shows a typical integration placement alongside identity controls.

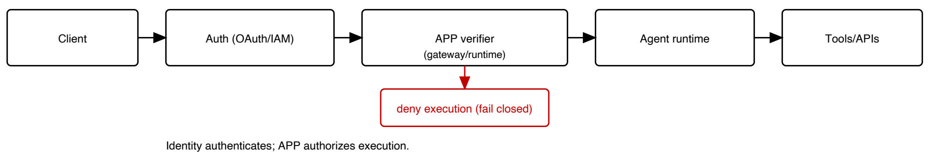
Client
Auth (OAuth/IAM)
APP verifier
(gateway/runtime)
Agent runtime
Tools/APIs
deny execution (fail closed)
Identity authenticates; APP authorizes execution.

Figure 6. Integration placement. APP complements identity systems by providing execution-time authorization at the runtime or gateway boundary.

11. Verifier compliance checklist

An APP-compliant verifier MUST:

- Deny unencrypted permission policies.
- Verify the signature before interpreting semantics.
- Validate required fields and supported policy_version.
- Enforce expiration strictly with a trusted clock.
- Enforce replay rules atomically when required.
- Enforce audience binding.
- Expose only allowlisted tools and capabilities.
- Fail closed on ambiguity, parse errors, or unknown fields.

12. Governance and standardization path

This proposal recommends:

- an open specification with versioned releases
- a conformance test suite for interoperability
- a reference validation checklist (MUST/SHOULD)
- community-driven review and evolution

Crittora proposes to steward the initial draft and contribute reference implementations to accelerate adoption.

Crittora is committing to open stewardship of APP while using it as the foundational authority layer across its own platforms.

13. Security considerations

Implementations must enforce fail-closed behavior, strict TTL checks, and audience binding. Replay protection must be atomic. Key management and trusted time are foundational assumptions.

14. Privacy considerations

Policies can include sensitive intent or metadata. Mandatory encryption protects confidentiality while preserving verifiability.

15. Conclusion

Agentic AI demands a new security boundary. APP defines explicit, time-bound, verifiable authority that can be enforced before any action occurs. This proposal invites the ecosystem to standardize agent authority as a protocol, making autonomous systems safer and auditable across platforms.