KINEGRANT PROTOCOL

KineGrant:
Verifiable Authorization for Physical Actions

KGP-001 Experimental Open Draft 0.1

Zoah

12 August 2026

PROTOCOL	IMPLEMENTATION	LICENSE
KGP-001 Draft 0.1	Reference v0.1.1	Apache-2.0

Abstract

Machines controlled by software increasingly cause state changes in the physical world. The software that decides what to do is often separated from the device that ultimately acts, while identity systems, transport security, robot middleware, and safety controls each protect a different boundary. This leaves a narrow but important question: what cryptographically verifiable authorization did this actuator possess for this exact action, at this target, at this time?

KineGrant is an experimental authorization and accountability protocol for that boundary. It converts a policy decision into a short-lived, signed, request-bound capability; places a fail-closed action gate in the actuator control path; consumes the capability atomically; and binds execution to a signed terminal receipt. The protocol is transport-independent, defaults to denial, applies deny-overrides, and prevents untrusted policy sources from expanding authority. This paper defines the system and threat models, decision semantics, capability construction, gate algorithm, receipt semantics, conditional security invariants, interoperability boundaries, and a reproducible software test method.

KineGrant does not solve identity enrollment, legal ownership, safe motion, sensor truth, key compromise, or actuator firmware compromise. A receipt proves an executor's signed attestation, not physical truth. The draft is intended to make the authorization boundary precise enough to implement, attack, reproduce, and revise.

Contents

1 Introduction

An AI system may infer an action, a robot application may transmit it, and a device may be technically capable of carrying it out. None of those facts establishes that the action was authorized. Intelligence is not permission.

Existing mechanisms provide essential controls: identity and access management authenticates principals; public-key infrastructure authenticates keys and endpoints; network controls limit connectivity; robot middleware governs communications; industrial protocols secure sessions; and functional-safety systems constrain hazardous behavior. These controls should remain in place[10, 11, 12]. The missing object addressed here is narrower: a device-verifiable grant that binds a specific request to a policy evaluation and can be consumed immediately before a physical action, followed by a signed result record.

KineGrant defines the following path. Its use of the term capability follows the security tradition of unforgeable authority-bearing references, while adding protocol-specific request, time, replay, and receipt semantics[13, 14].

The protocol contributes four coupled properties. First, authorization is explicit and bound to normalized request and policy digests. Second, permission is short-lived and single-use by default. Third, the gate is placed in the trusted path to the actuator and fails closed. Fourth, the outcome is represented by a signed, privacy-minimized terminal receipt.

The construction is not a claim that cryptography alone controls the physical world. Its security properties are conditional on trusted keys, time, policy provenance, canonical encoding, durable atomic replay state, and an actuator path that cannot bypass the gate. These conditions are part of the protocol's deployment contract, not implementation details to be assumed away.

2 Scope and Design Goals

2.1 Goals

KGP-001 is designed to provide:

1. **Exact binding.** A grant identifies one agent, target, action, purpose, request digest, and policy snapshot digest.
2. **Local verification.** A gate can decide without consulting a remote ledger or policy service at action time.
3. **Bounded authority.** A capability is short-lived, scoped, and consumed once by default.
4. **Monotonic restriction.** Untrusted policy and discovery inputs may restrict or deny but cannot create new permission.
5. **Accountability.** Execution produces a signed terminal attestation that can be checked independently against explicit trust anchors.
6. **Interoperability.** The authorization object can complement existing identity, middleware, device, and safety systems without replacing them.

2.2 Non-goals

KineGrant is not a robot operating system, motion planner, functional-safety controller, identity registry, legal authorization system, blockchain, token, or consensus network. It does not establish ownership, infer human intent, determine whether sensor data is true, or prove that a physical event occurred exactly as reported. It does not permit a software grant to override a native safety veto or emergency stop.

> A conforming deployment evaluates physical execution as a conjunction: native platform authorization AND a valid KineGrant capability AND the local safety controller must all allow. Any layer may veto. KineGrant cannot force another layer to permit.

3 System and Threat Models

3.1 Entities and trust boundaries

Entity	Role and trust condition
Agent	Requests an action. It may be buggy or compromised; its request is not permission.
Policy source	Supplies rules. An allow rule is effective only if its issuer is independently authenticated and explicitly trusted.
Policy engine	Evaluates normalized rules using trusted local time and produces a decision.
Authority	Signs capabilities. Its key and issuer identity are explicit gate trust anchors.
Action gate	Verifies, binds, and atomically consumes a capability in the actuator control path.
Replay store	Implements durable, linearizable consume-if-absent semantics for capability nonces.
Executor	Performs or supervises the action and signs one terminal receipt.
Verifier	Checks signatures, hashes, time, chain structure, and caller-supplied trust anchors.
Safety controller	Independently enforces motion, collision, process, and emergency constraints.

3.2 Adversary capabilities

The draft considers an adversary that can observe, delay, replay, reorder, and modify network messages; submit stale or future-dated requests; inject policy documents; exploit ambiguous adapters; attempt to substitute a target or action; race concurrent gate calls; restart a process; and tamper with an unanchored receipt log. A robot application may be compromised and may request or attempt a different action after receiving a capability.

The draft does not defeat theft of a trusted authority or executor key, compromised actuator firmware that bypasses the gate, a dishonest executor or sensor, denial of service, traffic analysis, physical coercion, or disputed legal authority. These remain explicit residual risks.

3.3 Assumptions

The arguments in Section 6 require: (A1) Ed25519 signatures are unforgeable for uncompromised keys; (A2) SHA-256 is collision resistant for the bound objects; (A3) the gate has an authenticated trusted-issuer set; (A4) policy trust is established outside policy content; (A5) gate time is sufficiently trustworthy; (A6) nonce consumption is durable and atomic; and (A7) every protected actuator command traverses the conforming gate. Violating an assumption invalidates the associated claim.

4 Protocol Objects

Let $H(x)$ be SHA-256 over the protocol's deterministic encoding of normalized object x. Let $\mathrm{Sig}_{sk}(x)$ denote an Ed25519 signature and $\mathrm{Verify}_{pk}(x, \sigma)$ its verification[3]. Draft 0.1 uses deterministic JSON; a later draft should adopt a standards-track canonical encoding such as JCS or deterministic CBOR[4]. Cross-implementation digest agreement is therefore a conformance requirement and an identified migration item.

4.1 Action request

An action request is the tuple

$$r = (rid, agent, target, action, purpose, t_{issued}, context).$$

The request identifier is globally unique. Identifiers and action names are normalized before evaluation. The request's timestamp is evidence supplied by the requester; it is not the clock used to decide policy validity.

The policy engine rejects requests older than the configured maximum age or unreasonably far in the future according to trusted local time.

4.2 Policy rule and normalized snapshot

A rule contains an identifier, issuer, effect in $\{allow, deny\}$, target pattern, subjects, actions, purposes, constraints, obligations, and optional validity window. The engine defaults to denial and applies deny-overrides. Unknown authorization fields, constraints, or operators fail closed. Missing target, subject, action, or purpose scope cannot be silently expanded into a wildcard.

For a request r, time t, rule set P, and independently configured trusted allow-issuer set T_P, define $Applicable(p, r, t)$ only when every normalized selector and every understood constraint matches. Define:

$$D(r, P, t) \equiv \exists p \in P : \ p.effect = deny \wedge Applicable(p, r, t),$$
$$A(r, P, T_P, t) \equiv \exists p \in P : \ p.effect = allow \wedge p.issuer \in T_P \wedge Applicable(p, r, t).$$

The decision function is

$$Outcome(r, P, T_P, t) = \begin{cases} deny & \text{if request freshness is invalid, parsing is incomplete, or } D(r, P, t), \\ allow & \text{if } A(r, P, T_P, t), \\ deny & \text{otherwise.} \end{cases}$$

This construction makes restriction monotonic with respect to untrusted sources: adding an untrusted document cannot turn a denial into an allowance.

The decision records $H(r)$, outcome, reason, matched policy identifiers, obligations, and $H(N(P, T_P))$, where N is the complete normalized policy snapshot together with the trusted issuer set used for evaluation. Binding the full snapshot prevents a policy-content change from retaining the same digest merely because a small subset of identifiers is unchanged.

4.3 Physical action capability

Only an allowed decision may produce a capability. Its signed body contains at least:

$$C = (v, cid, issuer, agent, target, action, purpose, H(r), H(N(P, T_P)), n, t_{nb}, t_{exp}).$$

Here n is a cryptographically random nonce, t_{nb} is not-before time, and t_{exp} is the exclusive expiry. The interval is half-open: $[t_{nb}, t_{exp})$. KGP-001 limits capability lifetime to 300 seconds and requires single-use consumption by default. Unknown versions, unknown fields, malformed base64url, or invalid time ordering are rejected.

4.4 Physical action receipt

A terminal receipt contains the executor identity, capability identifier, request digest, result, execution time interval, optional evidence hash, optional previous-receipt hash, and executor signature. The receipt is constructed from the gate's verified capability result, not from caller-reconstructed fields. A receipt log rejects a second terminal receipt for the same capability. Raw video, audio, precise location, or personal data should remain outside the receipt and may be referenced by a content hash when policy permits.

5 Authorization and Execution Protocol

5.1 Issuance

The policy engine evaluates request r against normalized rules and trusted local time. If the result is denial, no capability is issued. If allowed, the authority constructs C, chooses a fresh nonce, applies a lifetime no longer than 300 seconds, and signs the deterministic encoding. The authority does not make the decision self-authenticating: the gate separately authenticates the capability issuer against its own trust configuration.

5.2 Gate algorithm

The following abstract algorithm is normative in ordering where the order affects authority. An implementation may reject earlier for resource protection, but it must never invoke the actuator before all checks and successful atomic consumption.

```
AUTHORIZE_AND_CONSUME(signed_capability, observed_request, trusted_now):
  body, signature := STRICT_PARSE(signed_capability)
  require body.version == "0.1"
  require NO_UNKNOWN_FIELDS(body)
  require body.issuer in TRUSTED_CAPABILITY_ISSUERS
  require VERIFY_ED25519(body.issuer_key, body, signature)
  require body.not_before < body.expires_at
  require body.expires_at - body.not_before <= 300 seconds
  require body.not_before <= trusted_now < body.expires_at
  require body.request_digest == H(NORMALIZE(observed_request))
  require EXACT_MATCH(body, observed_request,
                      [agent, target, action, purpose])
  require VALID_POLICY_DIGEST(body.policy_digest)
  require ATOMIC_CONSUME_IF_ABSENT(body.nonce) == CONSUMED
  return VERIFIED_CAPABILITY(body)
```

If any requirement fails, the gate returns denial and does not call the protected actuator. The atomic operation must be linearizable for concurrent attempts and durable across the restart boundary claimed by the deployment. A memory-only replay store is appropriate for simulation, not a durable real device.

5.3 Execution and receipt

The executor receives the opaque verified result from the gate, performs the independently safe action if all native controls also allow, and signs one terminal receipt. The receipt result may record success, denial, failure, or interruption according to the protocol profile. Receipt verification checks deterministic encoding, signature, caller-controlled executor trust, capability binding, time consistency, evidence-hash syntax, previous hash, and terminal uniqueness.

6 Security Invariants and Analysis

The following are conditional protocol arguments, not a machine-checked proof, certification, or claim of immunity from implementation defects.

6.1 I1: no grant, no protected action

Under A3, A5, and A7, a protected actuator invocation can follow only a successful gate result. The gate returns success only after issuer, signature, lifetime, request binding, and replay checks. Therefore an absent, malformed, untrusted, expired, mismatched, or replayed capability cannot authorize the protected invocation. This invariant does not hold if another firmware or control path can reach the actuator without the gate.

6.2 I2: request non-substitutability

Under A1 and A2, changing agent, target, action, purpose, or any normalized request field changes $H(r)$ or the exact bound field. The adversary must then either forge the authority signature, find a digest collision, or cause divergent normalization. Strict schemas and conformance vectors reduce, but do not eliminate, the normalization risk.

6.3 I3: at-most-once authorization

Under A6, all successful uses of nonce n linearize at the consume-if-absent operation. At most one transition from absent to consumed exists. Consequently, concurrent, later, and post-restart replays are denied. This is an authorization property: an actuator or network fault could still make the physical result ambiguous, so idempotence and safe recovery remain device concerns.

6.4 I4: bounded validity

Under A5, the half-open interval check accepts C only when $t_{nb} \leq t < t_{exp}$ and $t_{exp} - t_{nb} \leq 300$ seconds. The exact instant $t = t_{exp}$ is denied. Requester-controlled timestamps cannot revive policy because policy windows and freshness are evaluated against the engine's trusted time.

6.5 I5: receipt integrity, not physical truth

Under A1 for an explicitly trusted executor, alteration of a signed receipt or its referenced hashes is detectable. A previous-receipt hash makes deletion or reordering detectable after a later trusted checkpoint is retained. Neither property establishes that the executor or sensor reported reality honestly. Independent attestation and observation are separate systems.

6.6 Attack analysis

Attempt	Required response	Residual condition
Forged capability	Reject invalid signature or untrusted issuer.	Trusted authority key theft is outside the software-only model.
Request mutation	Reject digest or exact field mismatch.	Divergent normalization across implementations must be tested.
Replay	Reject already-consumed nonce atomically.	Store must be durable and linearizable for the deployment boundary.
Concurrent replay	Exactly one consume operation may succeed.	Distributed stores require a documented consistency model.
Expired capability	Reject when $now \geq expires\_at$.	Clock rollback or compromise remains a deployment threat.
Policy injection	Untrusted issuer cannot produce allow; applicable deny may restrict.	Trust configuration compromise remains possible.
Backdated request	Evaluate policy time against trusted now; apply request-age limits.	Secure time synchronization remains necessary.
Ambiguous adapter	Reject unknown field, constraint, operator, or prohibition.	A known but incorrect mapping remains an adapter defect.
Receipt substitution	Verify expected capability, request digest, executor trust, and signature.	A trusted executor can still lie about physical reality.
Gate bypass	Deployment is non-conforming.	Requires hardware and control-path architecture, not a message-format fix.
Safety conflict	Native safety veto remains authoritative.	KineGrant is not a safety case.

7 Interoperability Boundaries

KineGrant complements existing systems by adding action-specific, short-lived authority at the local actuation boundary. It should reuse existing identities, keys, transports, and native authorization rather than duplicate them.

System	What it already provides	KineGrant boundary
IAM / PKI	Principal and key identity, authentication, trust distribution.	Supplies trust anchors; KineGrant binds them to one physical action and time window.
ROS 2 / SROS2	Enclave identity and middleware communication permissions.	A bridge gates the specific actuator command; middleware permission remains required.
OPC UA	Secure channels, application/user identity, roles, audit mechanisms.	A method adapter binds a capability to an exact node/method/action without weakening OPC UA controls.
Matter	Authenticated fabrics, device interaction, and message replay protection.	A target adapter adds protocol-level action authorization; Matter authorization remains authoritative.
W3C ODRL	Policy vocabulary for permissions, prohibitions, duties, and constraints.	A strict profile normalizes understood constructs and rejects unknown authorization semantics.
W3C WoT TD	Machine-readable affordances, forms, and security metadata.	Discovery may publish identity and policy pointers; unauthenticated discovery cannot grant.
Functional safety	Hazard controls, safe state, emergency handling, motion limits.	Always independent and veto-capable; never replaced by a capability.

The mappings use published platform and policy models as reference points, including ODRL and WoT Thing Description[8, 9]. Adapters are security boundaries. They must record source profile and transport, safely encode target components, reject caller attempts to overwrite system-owned context, and fail closed on unknown authorization constructs. Draft 0.1 does not claim conformance certification with the systems listed above.

8 Receipts, Privacy, and Auditability

An audit record is useful only if its claim is stated precisely. A KineGrant receipt supports the claim: "the holder of this executor key signed this terminal statement about the identified capability and request." With an explicit trusted-executor set, the verifier may additionally claim that the signer was trusted for the verification context. The receipt alone cannot support the stronger claim that a sensor was accurate or that a physical event occurred.

Privacy minimization is therefore structural. The base receipt carries identifiers, time, result, digests, and signatures. Rich evidence stays out of band. A hash can commit to external evidence, but collection, retention, disclosure, and deletion require independent policy and law. Receipt chains should be checkpointed outside the executor to make later deletion detectable; no public or consortium ledger is required in the real-time safety path.

9 Reference Implementation and Reproducible Evidence

The Apache-2.0 Python reference implementation is version 0.1.1; the wire protocol remains draft 0.1. It demonstrates Ed25519 capability and receipt signatures, default-deny policy evaluation, trusted policy and capability issuers, deterministic JSON digests, strict data models and JSON Schemas, in-memory and SQLite replay stores, adapters, a gate, and receipt verification.

Release v0.1.1 was validated with 33 automated protocol and security regression tests across the documented environment, a demo smoke test, Draft 2020-12 JSON Schema validation, and wheel/source package builds[7]. The separate Machine Permission Test (MPT) v0.1 publishes nine machine-readable cases:

1. no capability: deny;
2. valid capability: exactly one authorization;
3. replay: deny;

4. mutation of agent, target, action, or purpose: deny;
5. wrong capability issuer: deny;
6. expired capability: deny;
7. concurrent use: exactly one success;
8. restart followed by replay: deny;
9. receipt tampering or untrusted executor: deny.

The reference evidence reports 9/9 PASS. This is project-generated software evidence, not an independent audit, formal proof, physical-device validation, or safety certification. A useful external reproduction must record exact repository commit, runtime, operating system, dependencies, command, raw evidence, and cryptographic hashes; failures and deviations should be published rather than summarized away.

```
git clone https://github.com/zoahdev/kinegrant-protocol.git
cd kinegrant-protocol
python -m venv .venv
# activate the environment, then:
python -m pip install -e '.[test]'
kinegrant-mpt --output machine-permission-test.evidence.json
python -m unittest discover -s tests -v
```

The authoritative commands, checksums, release assets, and platform-specific instructions are maintained in the repository and release notes. Reproduction establishes behavior in the tested software environment only.

10 Deployment Requirements

A real deployment claiming KGP-001 behavior should document and review at least the following:

1. place the gate in the mandatory actuator control path;
2. keep native authorization and an independent functional-safety controller enabled;
3. store authority and executor keys in hardware-backed keystores where available;
4. provision issuer trust and revocation independently of untrusted input;
5. use monotonic or securely synchronized time and define clock-failure behavior;
6. provide durable, atomic nonce consumption across the stated restart and concurrency scope;
7. pin adapter profiles and reject unknown or ambiguous constructs;
8. minimize receipt data and checkpoint receipt-chain state outside the executor;
9. fail safe when policy, adapter, registry, time, or replay services are unavailable;
10. complete independent cryptographic, software, privacy, and machinery-safety review.

The protocol should not be the sole control for hazardous machinery. A deployment report should identify bypass paths, trusted computing base, key lifecycle, update mechanism, storage failure modes, time source, action recovery semantics, and test evidence.

11 Open Problems

Draft 0.1 intentionally leaves several issues unresolved: conflicts among legitimate co-owners; revocation during a running action; delegation and attenuation; multi-agent actions; privacy- preserving discovery; emergency and accessibility override semantics; hardware-backed actuator attestation; distributed replay-store consistency; cross-language canonicalization; long-running actions whose duration exceeds grant lifetime; and formal verification of policy adapters and gate implementations.

These are not roadmap decorations. Each affects the meaning of authorization and should be resolved through explicit threat models, test vectors, incompatible-change rules, and public review before a stable protocol claim.

12 Conclusion

Physical AI creates a boundary where an intention represented in software becomes an action in the world. KineGrant isolates one question at that boundary: did this actuator receive a valid, specific, short-lived, non-replayable authorization for this request, and can the executor's terminal statement be verified later?

KGP-001 answers with a policy decision, signed capability, mandatory local gate, atomic consumption, and signed receipt. Its value depends less on the novelty of any cryptographic primitive than on composing the trust boundaries without silently widening authority. The draft therefore favors explicit trust, exact binding, default denial, short lifetimes, local verification, and falsifiable evidence. Its current status remains an experimental open draft: suitable for implementation, attack, and reproduction, not for unsupported claims of safety or standardization.

Protocol Status and Resources

Protocol	KGP-001 Experimental Open Draft 0.1
Reference implementation	v0.1.1
License	Apache-2.0
Repository	`https://github.com/zoahdev/kinegrant-protocol`
Protocol page	`https://kinegrant.com/paper`
Machine Permission Test	`https://kinegrant.com/challenge`
Issue reporting	`https://github.com/zoahdev/kinegrant-protocol/issues`

References

[1] KineGrant Protocol, "KGP-001: KineGrant Core Authorization Protocol," Experimental Open Draft 0.1, 2026. `https://github.com/zoahdev/kinegrant-protocol/blob/main/spec/KGP-001.md`

[2] KineGrant Protocol, "Threat Model," 2026. `https://github.com/zoahdev/kinegrant-protocol/blob/main/spec/THREAT-MODEL.md`

[3] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, 2017. `https://www.rfc-editor.org/rfc/rfc8032`

[4] A. Rundgren, B. Jordan, and S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, 2020. `https://www.rfc-editor.org/rfc/rfc8785`

[5] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, 1997. `https://www.rfc-editor.org/rfc/rfc2119`

[6] B. Leiba, "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," BCP 14, RFC 8174, 2017. `https://www.rfc-editor.org/rfc/rfc8174`

[7] JSON Schema, "Draft 2020-12," 2022. `https://json-schema.org/draft/2020-12/`

[8] W3C, "ODRL Information Model 2.2," W3C Recommendation, 2018. `https://www.w3.org/TR/odrl-model/`

[9] W3C, "Web of Things (WoT) Thing Description 1.1," W3C Recommendation, 2023. `https://www.w3.org/TR/wot-thing-description11/`

[10] Open Robotics, "ROS 2 Security: Understanding the Security Keystore." `https://docs.ros.org/en/ros2_documentation/jazzy/Tutorials/Advanced/Security/The-Keystore.html`

[11] OPC Foundation, "OPC Unified Architecture - Part 2: Security Model." `https://reference.opcfoundation.org/specs/OPC-10000-2/`

[12] Connectivity Standards Alliance, "Matter Core Specification 1.0," 2022. `https://csa-iot.org/wp-content/uploads/2022/11/22-27349-001_Matter-1.0-Core-Specification.pdf`

[13] J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, 1966. https://doi.org/10.1145/365230.365252

[14] M. S. Miller, K.-P. Yee, and J. Shapiro, "Capability Myths Demolished," Systems Research Laboratory Technical Report SRL2003-02, 2003. https://srl.cs.jhu.edu/pubs/SRL2003-02.pdf

A Conformance Checklist

The following checklist is informative but maps directly to normative draft requirements.

ID	Requirement	Evidence
C1	Default deny and deny-overrides are enforced.	Test vector / code
C2	Allow issuers are authenticated independently and explicitly trusted.	Configuration / test
C3	Untrusted policy or discovery input cannot create allowance.	Negative test
C4	Unknown authorization constructs and fields fail closed.	Parser tests
C5	Policy windows use trusted engine time; request freshness is bounded.	Clock tests
C6	Capability binds exact request fields, request digest, and full policy digest.	Mutation tests
C7	Capability issuer, signature, version, fields, and base64url are strictly checked.	Negative tests
C8	Validity is half-open and no longer than 300 seconds.	Boundary tests
C9	Consumption is atomic, one-time, and durable for the claimed restart scope.	Race / restart tests
C10	The protected actuator has no path that bypasses the gate.	Architecture review
C11	Receipt is built from a verified capability and has one terminal result.	API / log tests
C12	Receipt trust uses caller-controlled trusted executors.	Trust test
C13	Native safety logic remains independent and veto-capable.	Safety architecture
C14	Limitations, key lifecycle, time, storage, and recovery semantics are documented.	Deployment report

B Requirement Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in the normative KGP-001 specification are to be interpreted as described in BCP 14 (RFC 2119 and RFC 8174) when, and only when, they appear in all capitals.

NO GRANT. NO ACTION. VERIFIABLE RECEIPT.