Approval You Can Verify: Owner-Approved Agent Spending with Sealed Evidence

Sai Arun Gunuganti

Sealgraf — Contact: support@sealgraf.com

Technical paper, version 1.2, 15 November 2026. This paper is not peer-reviewed. It is published as a technical description of a running system, as prior art, and as a contribution to protocol work in progress. The system it describes, Sealgraf, has been in production since 2 November 2026.

License: Creative Commons Attribution 4.0 International (CC BY 4.0). Canonical text: Sealgraf. This PDF was rendered from the canonical Markdown source (paper.md) on 15 November 2026.

Abstract

When a software agent spends a person's money, someone must be able to answer, later and to a third party: *what exactly did the owner approve, under what limits, and what happened next?* Current agent protocols answer the first part of that question with a statement by the agent itself. The Personal Agent Protocol's Operations extension states the consequence in its own security section: an approval there "doesn't prove that the User saw the terms or approved them" — it "works on trust." This paper describes a system, running in production, that removes the trust step for the approval half of the problem. An owner sets machine-checkable limits in advance (a *mandate*). An agent checks each proposed purchase against those limits before spending; a deterministic engine decides, and the decision is persisted and sealed before any spendable token exists. Purchases above an owner-set line are decided by the owner personally — or by a person the owner invited to approve — on their own authenticated surface, a surface the agent can notify but cannot operate, imitate, or replay. Approval mints a single-use token bound to the hash of the exact action approved. The result is a sealed, owner-held receipt in which two grades of fact are kept deliberately separate: the approval is *proven*; the performer's report of what happened next, including the payment details, is *reported and pinned*. The paper states the design, the evidence model, the operating record so far (including its limits), the relation to neighbouring work — the Personal Agent Protocol, AP2, Mastercard Verifiable Intent, Asqav, the Izmaylov permission-receipts draft, and x402 — the open problems, and a concrete proposal: an approval-evidence object that protocols such as the Operations extension could carry, so that a company which needs more than trust can verify an approval mechanically.

1. The problem: approval attested by the party it constrains

Agent spending separates the person who owns the money from the actor who spends it. Between them sits an approval: some act by which the owner, or the owner's standing instructions, authorized *this* purchase. Every dispute, refund claim, and audit in this space reduces to three questions about that approval:

1. What were the exact terms approved — item, amount, counterparty?
2. Who approved them, and how do we know?
3. What happened afterward, and who says so?

The protocols now being drafted answer question 1 well and question 2 hardly at all. The Personal Agent Protocol (PAP, "Poppy"), announced by Sierra and Meta on 6 October 2026, defines an Operations extension in which a company proposes an action with exact, immutable terms (a *revision*), the user's personal agent obtains the

user's approval, and the company performs the action at most once. The binding of terms is careful work. The approval, however, is a statement by the agent: the agent sends the operation's identifier and revision back, and the company accepts the agent's word that the user approved. The extension's security section (§8.1 of Operations version 2, protocol draft 0.3) says so itself:

> "It doesn't prove that the User saw the terms or approved them. The Company can't see the Personal Agent's interface, so an approval works on trust, like the sign-in types. A Company MAY revoke the `client_id` of a Personal Agent that sends an approval the User didn't give. A Company that needs proof of the User's approval for an action can ask the User to approve on its own website, signed in, the way Direct Sign-In works. This extension doesn't define that step. Approvals the User signs are an open topic."

Three properties of this arrangement deserve emphasis, because they are not specific to PAP — PAP is simply the design that states them honestly:

- **The attestor is the constrained party.** The agent is the actor whose behaviour the approval is supposed to constrain. An approval channel the agent itself speaks through cannot distinguish a real approval from a fabricated one; it can only punish fabrication after the fact (PAP's remedy is revocation of the agent's `client_id`).
- **The approval surface is the agent's interface.** Whatever the user saw and tapped happened inside software the agent controls. The company cannot see it; neither, later, can an arbitrator.
- **The record belongs to someone else.** Operations gives the two parties a shared operation record held by the company. The owner — the person whose money moved — holds no artifact of their own: nothing signed, nothing they can present to a third party who trusts neither the agent nor the company.

The same hole appears wherever an agent's self-report stands in for its principal's consent. Standing permissions make it wider: PAP Operations version 1 permitted a confirmation to cite a `standing_permission` — the user's earlier blanket permission, in free text ("exchange for a different size if it costs under $20") — with no machine-checkable form and no way for the company to verify that the permission covered the terms. Version 2 withdrew the delegated form (the agent "MUST NOT approve one on the User's behalf, even when the User said earlier that they would approve something like it"), which solves the fabrication problem by removing delegation rather than by evidencing it. Delegation is, however, the normal case for a spending agent: an owner who must personally approve every purchase has a notification service, not an agent.

The position taken by this paper is that the approval half of agent spending is an *evidence* problem with a known shape: move the approval off the agent's surface, bind it to the exact terms, record it before anything spendable exists, and give the resulting artifact to the owner. Section 2 describes a production system built on those

four moves. Section 5 is explicit about what neighbouring systems do differently or better; nothing in this paper claims the pieces are new. The claim is narrower: the combination is running, and the full loop has executed end-to-end with real money in the operator's own deployment (Section 4, including its limits).

The mechanism in this paper exists for a reason larger than the disputes it settles. As agents begin to spend, shop, and commit on a person's behalf, the default outcome — unless it is designed against — is that the person quietly stops being the owner of their money in any practical sense: the agent acts, the platform attests, and the owner's part is reduced to having once agreed to terms they never saw. The design position of this paper is the opposite. The owner sets the terms in advance, decides personally above a line they chose, and holds the evidence themselves. Delegation does not have to mean surrender: an agent can be given real power to act while the authority over the money — and the proof of how it was used — stays with the person whose money it is. That, and not the receipt format, is the claim the mechanism serves.

2. Design

Sealgraf is an approval-and-evidence layer for agent spending. It checks; it never moves money. An agent asks before it spends; the owner's money moves on whatever rail the agent and merchant use (a wallet, a card, PayPal), untouched by Sealgraf. What Sealgraf produces is the record: what was asked, what was decided, by whom, under which limits, and what the agent reported afterward.

2.1 The mandate: standing permission as data

An owner's standing permission is a **mandate**. A mandate is a sequence of immutable versions; every change the owner makes creates a new version, and every decision cites the exact version it was judged under. One version's terms carry, as fields:

- `agent_id` — the one agent the permission covers;
- `categories` — an allowlist of purchase categories;
- `per_purchase_cap_minor` — the most any single purchase may cost;
- `period_cap_minor` and `period_days` — the most that may be spent in a rolling period;
- `currency`;
- `not_before` / `expires_at` — the validity window;
- `human_approval_threshold_minor` — the **ask line**: at or below it, the agent may proceed on the standing permission; above it, the owner decides that purchase personally;
- `status` — active, paused, or revoked.

Mandates are owner-set and never agent-editable. This is PAP Operations v1's free-text standing permission written as data — with one structural addition that has no PAP counterpart: the ask line keeps a per-purchase human decision available *inside* a standing permission, so delegation is a gradient rather than an all-or-nothing switch.

2.2 The check: deterministic, fail-closed, recorded first

Before spending, the agent submits a proposed action — agent, category, amount, currency, counterparty, description — to the check (over MCP as the `check_permission` tool, or over REST). The SHA-256 of the action's canonical bytes is its **action hash**; everything downstream — decision, approval, token, receipt — cites it, so the action approved and the action later reported can be compared by any third party without trusting either side's summary.

The decision is made by a deterministic rule engine: a pure function of (mandate version, period spend so far, proposed action, time). No model is anywhere in the decision path. Rules fire in a fixed order, and every outcome carries a stable reason code:

1. agent mismatch (`agent_mismatch`) — the action names a different agent than the mandate covers;
2. mandate not active, not yet valid, or expired;
3. scope: currency mismatch, non-positive amount, category not in the allowlist (`category_not_allowed`), amount over the per-purchase cap (`over_per_purchase_cap`);
4. period cap: completed plus reserved spend plus this amount over the period cap (`over_period_cap`);
5. ask line: amount over the threshold escalates to the owner (`over_human_approval_threshold`).

Anything else is allowed. The failure direction is closed: **the decision is persisted to the ledger before any decision token exists**. If persistence fails, no token is issued and nothing can be spent against the check. Authorization also *reserves* against the period cap — the engine counts completed plus outstanding unexpired reservations, computed and persisted inside one transaction serialized per mandate — so two concurrent checks cannot both spend the same headroom, and an approval re-validates the reservation before anything is minted.

2.3 The approval: on a surface the agent cannot write

When a check escalates, the owner is notified — by email or push — and decides on their own authenticated surface (the Sealgraf web app, signed in as the owner). The notification is a **doorbell**: it carries the facts and a link, never the decision control. No approval can be given from an email, a push notification, or any channel the agent can originate; the Allow and Don't-allow controls exist only inside the owner's

authenticated session. This is the property the whole design turns on: what makes the approval hard to fabricate is not the signature applied to it later, but that the decision happens where the agent cannot write.

The approval is recorded as its own sealed statement, carrying:

- the action hash approved;
- the authenticated principal who approved — the account, recorded as authenticated, never a free-text label (refusals and aborts record the principal the same way);
- the mandate version the approval was judged against — re-validated at approval time against the *current* version (status, window, scope, period headroom), not merely the version in force when the agent first asked;
- the approval time, and — for approvals at or above an owner-set step-up line — a record of the authentication context (how recently and by what method the approver authenticated). The verifier displays this freshness record and applies no verdict to it; whether it is good enough is the reader's policy.

Invited approvers. The owner may invite up to three people — a partner, a bookkeeper — to decide escalations too. An invited approver decides the same way the owner does: in their own authenticated session, never from a notification or any channel the agent can originate. They hold no other authority — they cannot change the mandate, the keys, or any setting — and the owner can remove them at any time, effective on their next action. Their approval is sealed in its own form (approval `schema_version` 5): the principal is the approver's *own* account, in the role `approver`, so a receipt never presents an approver's decision as the owner's. Owner approvals keep the earlier form unchanged, and a verifier that predates version 5 rejects an approver's receipt as unsupported rather than misreading it.

Approval mints a **decision token**: single-use, bound to the action hash, valid for 15 minutes (900 seconds), stored only as a hash. The token is escrowed encrypted for exactly the credential that ran the check, which may **claim** it after the decision — claim is not consumption — so an agent that was offline when the owner decided can still collect the token, spend, and complete when it returns, with no new check and no broader access. (In operation, agents arm a claim poller, polling roughly every 30 seconds after firing an ask, so the owner's tap alone drives the rest of the flow.)

Approval is not the last check. At completion, the engine re-validates the token, its expiry, and the mandate's *current* status and scope under the same per-mandate lock: a pause, revoke, or tightening revision that lands between approval and spending stops the spend.

Two further pieces of the approval surface matter in operation:

- **Refusals are first-class.** If the owner refuses a purchase from the pending state, the refusal seals — an abort entry symmetric to an approval, naming the aborting principal, the state aborted from, and the mandate version in force. An unchanged re-check of the same action is then denied with the reason `action_previously_denied`, keyed by the action hash and deliberately surviving later mandate revisions: editing the limits does not resurrect a refused purchase.
- **The ask channel itself is shaped.** An agent that asks too often is itself a failure mode for the owner. Repeated asks and repeated refusals within a window produce shaped denials — sealed decisions with their own reason codes (`ask_budget_exhausted`, `ask_cooling_off`) — which mint no token and reserve nothing, but are recorded like any other decision. The owner's brake always works; only the agent's future asking is shaped.

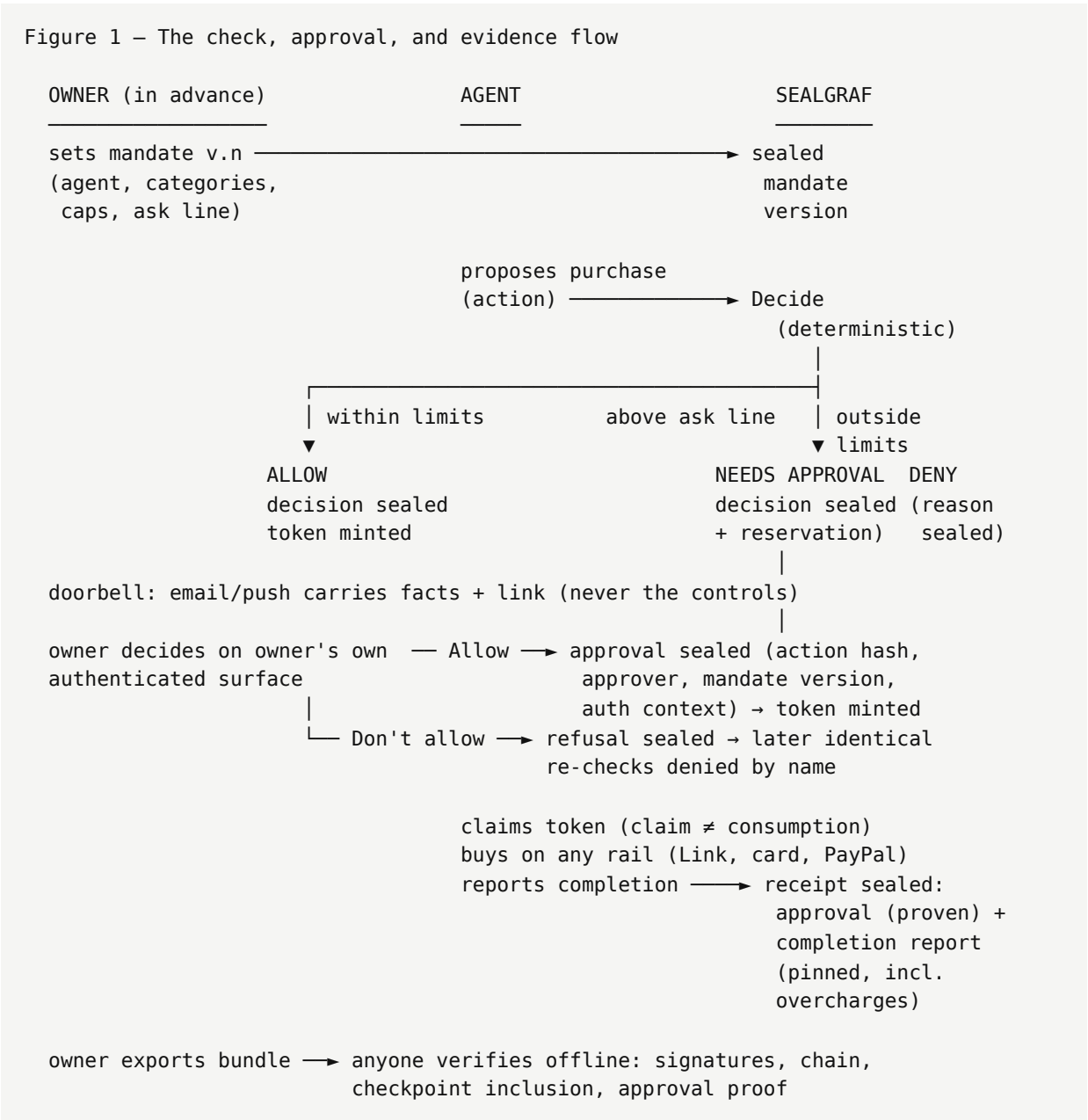
2.4 Completion: the performer's report, pinned

After spending, the agent reports completion: the completed amount, the merchant's reference, and optionally a payment block — rail (a closed vocabulary: Link/Stripe, PayPal, Stripe API, card on merchant, other), instrument label and last four digits, the one-time virtual card's last four digits as a distinct field, the charged amount and currency, the rail's own reference, and the report's provenance (how the agent learned the facts: browser checkout, rail API, or merchant confirmation). The report is validated for shape and screened (a full card number anywhere in it refuses the completion), then sealed exactly as reported.

One rule is load-bearing: **a completion is sealed even when it reports a charge above the authorized amount.** Refusing it would erase the only durable record of the overcharge. The comparison belongs at read time: every read of the receipt re-derives an overcharge flag from the sealed bytes — a payment block present, in the action's currency, charged above the amount that was checked and allowed. A different currency derives no flag and no conversion; the record shows both figures and the owner judges.

Completion is atomic: the ledger append, the receipt insert, and the preauthorization's conditional state transition happen in one store transaction, with a storage-level uniqueness constraint making a second receipt for one preauthorization impossible.

Figure 1 shows the flow end to end.



2.5 Sealing, custody, and verification

Every act in the flow — mandate version, decision, approval, refusal, abort, completion — is appended to a per-owner (per-tenant) hash chain: each entry's hash covers the previous entry's hash and the exact payload bytes, the payloads are stored as the bytes that were signed, and every append first verifies the current chain head, so a corrupted head is refused rather than sealed over. Entries are in-toto-shaped statements carried in DSSE envelopes, each envelope dual-signed (Ed25519 and ECDSA P-256; the P-256 key is held in AWS KMS and is non-exportable), and signing covers the DSSE pre-auth encoding of the exact payload bytes — verification never re-serializes a payload to check it.

Periodically, the chain is checkpointed: a Merkle tree head over the entries is dual-signed, and the checkpoint digest — the only bytes that leave the operator — can be witnessed externally (in the current deployment, by two independent RFC 3161 timestamp authorities; OpenTimestamps witnessing is supported by the same machinery). Checkpoints are currently volume-triggered, which Section 6 names as an open problem.

The custody point is the design's centre of gravity: **the receipt is the owner's artifact.** A receipt is one sealed bundle holding the cited mandate version's terms, the action and its hash, the decision, the approval statement when one was needed, and the completion block. Bundles export into an offline package — exact signed bytes, public keys, chain position, Merkle inclusion proof into a dual-signed checkpoint — that a verifier checks with **no account, no API key, and no trust in Sealgraf's servers**. The online checker applies the same discipline in the other direction: a receipt number alone opens nothing (it is a label anyone may photograph, not a credential); the receipt's check code, which only the owner and whoever they show it to possess, opens the full record.

The verifier keeps two verdicts separate, and the separation is part of the format: **authenticity** (the bytes are what the signer sealed and have not mutated) and **currency** (whether the cited mandate is still in force now — which offline verification cannot check and says so plainly). An approval inside a bundle is *proven* only when its entry is Merkle-included in a checkpoint that itself verified; chain-linkage alone is arithmetic a presenter could recompute over a fabricated statement, so an unanchored approval is reported as exactly that — `unanchored` — and a bundle whose receipt required approval but carries no approval statement fails verification outright. Approval can never be silently absent from the package that claims it happened.

3. The evidence grades

The design's central discipline is that a receipt does not make one undifferentiated claim. It carries facts of two different grades, and the grades are part of the artifact — stated on its face, enforced by its verifier — not a disclaimer attached to it.

Grade 1 — proven. The approval: that this owner, authenticated, approved this exact action (this action hash), under this version of their limits, at this time. This is provable because of where it happened (a surface the agent cannot write), when it was recorded (before any token existed), and how it is anchored (sealed, chained, checkpoint-included). The mandate itself, the decision, refusals, and aborts carry the same grade: they are the service's sealed records of acts by identified principals, verifiable offline.

Grade 2 — reported and pinned. Everything the performer says about what happened next: the completed amount, the merchant's reference, and the payment block — rail, instrument last-4, virtual card last-4, charged amount, rail reference. Sealgraf never touched the money and cannot prove a charge; what it can do is fix the report at the moment it was made, so neither side can edit it afterward, and grade it honestly. A verifier reading the bundle sees both grades and is never invited to mistake one for the other.

Figure 2 — Evidence grades inside one receipt

Fact	Grade	Basis
Owner set these limits (version n)	PROVEN	sealed mandate statement, owner-authenticated act
Engine allowed / escalated / denied (reason code)	PROVEN	decision persisted before any token exists
Owner approved this action hash at this time, under version n	PROVEN	approval on owner's own surface, sealed, anchor-included in a checkpoint
Owner refused; re-checks denied	PROVEN	sealed refusal (abort entry)
Agent completed the purchase; charged $X on rail R, card ••1234 (incl. charges above the amount approved)	REPORTED + PINNED	agent's completion report, validated for shape, sealed as reported; overcharge flagged at read time, never erased

Why grade at all? Because the alternative — one claim at the strength of its weakest part — is how approval evidence loses credibility. Inside flows where a payment protocol's merchant and processor both participate, the payment leg can carry a counterparty signature, which is stronger than a report (AP2's, Section 5.2, is exactly this, and this design concedes the point). But most agent purchases happen at ordinary checkouts where no counterparty signs anything for the owner. For those, the attainable grades are a proven approval and a pinned report, and an artifact that labels them correctly is worth more in a dispute than an artifact that blurs them into "verified."

4. Operating evidence

The system described above is not a design exercise. It has been in production since 2 November 2026, and the full loop — agent check, owner approval, automatic token claim, purchase on a real rail, sealed receipt — has run end-to-end with real money:

- **2 November 2026, leg 1.** An agent checked a $10.27 purchase (PayPal rail) at a national home-improvement retailer; it proceeded under the standing mandate and sealed receipt № 000019.
- **2 November 2026, leg 2.** An agent checked a $9.72 purchase paid with a Stripe Link one-time virtual Visa; the sealed receipt (№ 000025) carries the payment block — rail, the virtual card's last four digits, the charged amount. The same session produced the estimate lesson that shaped the design: the wallet's approval card had displayed an estimated total of $14.00 against the $9.72 request, and no artifact on the buyer's side kept that figure — the estimate existed only on a phone screen. (An early version of the payment block carried the displayed estimate as a sealed field; it was removed by a later ruling when its collection proved unreliable across rails — the comparison function survives in the read-time overcharge flag. The removal is recorded here because the episode is the design's origin, not its current state.)
- **2 November 2026, denial memory.** An identical re-check of a purchase the owner had declined was denied by name — `action_previously_denied`: in one production instance, a refusal bound a later presentation of the same action, not only in tests.
- **3 November 2026, the showcase loop.** An agent asked to buy a $9.98 item at the same retailer. The amount sat above the owner's ask line, so the check escalated; the owner approved from their phone; the agent's claim poller collected the decision token automatically, with no further prompting; the purchase completed through Stripe Link (charged $9.72 on a one-time virtual Visa, as reported) and sealed as receipt № 000030 — check code `a424e427cfcc419c`, publicly checkable by anyone the owner shows it to.

Production state as of 7 November 2026: 257 accounts, 216 tenants, 1,868 ledger entries, 103 sealed receipts, 26 agent connections.

The caveat belongs in the text, not a footnote: **almost all of those accounts are the operator's own test accounts, and the number of external users is zero.** The operating evidence therefore supports claims about the *mechanism* — that the loop runs, that approvals bind, that refusals hold, that receipts verify — and no claims about adoption, usability at scale, or owner behaviour in the wild. The mechanism claims are also, so far, claims about one operator's deployment, one agent ecosystem (Muse, via MCP), and US-dollar card and wallet rails. The paper's remaining sections should be read with that scope in mind.

5. Relation to other work

5.1 The Personal Agent Protocol (PAP / "Poppy")

PAP standardizes who the agent is, how it signs in, which doors a company opens for it, and the propose → approve → perform-once handshake of its Operations extension. The mapping between Operations and the system described here is close enough to state term by term:

- Operations v1's `standing_permission` (free text) ↔ a mandate version (the same permission as machine-checkable data; Section 2.1). Operations v2 removed the delegated form; the same concept has reappeared in the protocol's payments discussion (issue #61) as the intent mandate carrying user-approved constraints — the mandate maps to both seats.
- An operation (`operation_id`, `revision`, `terms`) ↔ a checked action. The direction is reversed — in Operations the company proposes; here the agent proposes and the check judges — but the binding device is the same idea in a different shape: PAP's revision identifies terms the company wrote; the action hash identifies terms the *agent* wrote, which is what lets a third party confirm the action approved is the action performed.
- A confirmation (`approved_by`) ↔ owner approval plus a single-use decision token (Section 2.3). A v1 confirmation is a statement by the agent about the user; an approval here is an act by the owner, recorded as its own sealed statement.
- `user_approval_required` ↔ the ask line — the same lever ("this one needs a human") held by different hands: the company sets it per operation in Operations; the owner sets it in advance, per mandate, here. A system with both would be stronger than either.
- The operation record and `result.summary` ↔ the sealed receipt. Operations' record is the company's account, held by the company; the receipt is the owner's counterpart, held by the owner — and its completion block keeps the two evidence grades separate in exactly the place Operations' framing invites merging them (Section 3).

What does not map: Sealgraf implements no PAP discovery, identity, session, or sign-in, and claims no PAP conformance. PAP operations cover any business action; this system covers purchases only. And PAP has no amounts vocabulary at all — its limits are scopes, which doors are open, never how much may be spent through them.

5.2 AP2 (Google, now at the FIDO Alliance)

AP2 is the strongest neighbouring design on the payment leg, and the concessions come first. Its mandates are user-signed, amount-bounded credentials verified on a non-LLM trusted surface — the same answer to the §8 hole, reached independently. Version 0.2 (April 2026) requires a merchant-signed Checkout Receipt and a

processor-signed Payment Receipt, hash-bound to the mandate and verified together as dispute evidence: *inside a flow where merchant and processor both run AP2, its payment leg is counterparty-signed, which is stronger evidence than an agent's report.* The differences are of position, not concept. AP2's chain exists only where both counterparties participate; the purchases in Section 4 happened at an ordinary browser checkout where neither did, and no AP2 artifact could have been produced. AP2's records are held by the flow's parties and verified between them; the receipt here is held by the owner and verified offline, by the owner, without any party's permission. And AP2's receipts are counterparty accounts of the payment; the approval in this design is the owner's own act, proven at the owner's side — the two artifacts answer different dispute questions and would compose rather than compete.

5.3 Mastercard Verifiable Intent

Verifiable Intent (v0.1 draft, March 2026; pilots from February 2026) records the consumer's request as a signed Intent Artifact that later transactions reference — in Mastercard's terms, a tamper-resistant record of what a consumer authorized an agent to do — consultable by issuers, merchants, and consumers at dispute time. Three structural differences: the artifact captures intent *before* the transaction and the network checks the transaction against it, rather than sealing the payment facts into an owner artifact after the fact; custody is the network's infrastructure — the parties *consult* the record, nobody *holds* it or verifies it offline; and it exists only on Mastercard rails with a participating issuer. Agent Pay's network-enforced budgets are enforcement this design does not attempt: Sealgraf checks and records; it never sits in the authorization path of a charge.

5.4 Asqav

Asqav ships the closest verification posture: signed action receipts (ML-DSA, post-quantum — cryptographically ahead of the Ed25519 / ECDSA P-256 used here), RFC 3161 timestamps, an open verifier, bilateral receipts that cryptographically link a pre-execution approval to a post-execution outcome, and an IETF-drafted receipt profile for regulatory compliance records. The boundary is the domain and the principal. Asqav signs *actions* for organizations: its outcome half is an agent-reported result digest of whatever a tool call did; its human approver is an organization reviewer in a dashboard queue, and its own documentation is candid that creating or deciding an approval "does not itself pause or resume agent execution" — the gating is assembled by the integrating developer, and its terms of service exclude consumers. Nothing in its format models a payment: no rail, no instrument, no charged amount, no rail reference, no charged-vs-authorized comparison. The systems share a family resemblance — sealed, verifiable receipts — aimed at different humans: an organization governing its agents, versus a person governing theirs.

5.5 The Izmaylov draft (draft-izmaylov-agent-permission-receipts)

The closest standards-layer neighbour, and the one this paper most needs to distinguish rather than cite past. Izmaylov's draft (-02, 31 October 2026) specifies records with which a third party can establish, later and without network access, what an agent was permitted to do and what it recorded doing: the principal signs a permission *with a passkey*, carrying limits a verifier can check mechanically; the agent signs a receipt per action (Ed25519 and ML-DSA-87); receipts are chained into a log whose signed tree heads are timestamped; and the verifier reports soundness and within-permission as separate answers — a separation this paper's Section 3 endorses and extends to the approval/completion boundary.

Two differences are structural. First, **signature versus surface**: the draft's principal signs the permission with their own key — a stronger cryptographic binding of the owner to the permission than this system's sealed record of an owner-authenticated act, and one the author of this paper regards as the better long-term shape for the mandate itself. What the draft does not specify is the per-action approval moment for a purchase above a line: its permission's conditions can require "the principal's approval of one action," but the ceremony — where that approval happens, on a surface the agent cannot operate, recorded before anything spendable exists — is outside its scope. Second, **evidence versus enforcement**: the draft states plainly that its records are "evidence, not enforcement," names the role of an *executor* that refuses an action its check does not allow, and expressly does not specify how an executor decides or keeps its records. The system described here is, in the draft's own vocabulary, an executor with a hosted owner surface and a payment block — plus the per-action approval ceremony. The formats also differ (JWS over canonical JSON there; DSSE envelopes here), which is a reason the two lines of work should converge on shared semantics even if encodings differ.

5.6 x402 (Coinbase)

x402 collapses the payment leg and its proof into one act: the payer's signed authorization *is* the payment instruction, and settlement is a public-ledger fact anyone can verify — the strongest payment proof in this space, for the payments it touches. It does not carry the owner's prior permission as an artifact bound to the settlement (wallet caps and session keys bound what an agent *can* sign; the protocol treats budget management as out of scope), it covers stablecoins on-chain and no fiat rails, and it proves payment, not delivery. The complementarity is exact: x402 is strongest where this design's payment evidence is weakest (settlement proof), and silent where this design concentrates (the owner's permission and approval, on the rails people actually use for purchases).

6. Open problems

Interception honesty. Sealgraf does not intercept anything. The agent is built or configured to call the check first; an agent that never calls it spends with no receipt, and the system does not detect the omission. (Asqav's documentation states the identical limit for its own governed path.) The design's answer is evidentiary rather than architectural: a spend *with* a check produces owner-held proof, and its absence is itself legible to the owner — but legibility is not prevention, and any deployment claim that blurs this is false. Closing the gap requires the check to sit in a path the agent cannot route around. That integration is not built.

Checkpoint cadence. Checkpoints are volume-triggered. For a low-volume owner, that means the newest entries — often the ones in dispute — can sit un-checkpointed for a long time, and an approval among them verifies only as `unanchored` until a checkpoint covers it. Volume is the wrong trigger; nobody serious anchors on a count. The direction under consideration is a signed tree head on every sealing event with external anchoring on a stated clock, using the existing OpenTimestamps / dual-TSA machinery. It is a small change with a large evidence consequence, and it is deliberately not folded into this paper's claims until it ships.

The test-check problem. A connection that has never fired a real check is indistinguishable from a broken one, which is why the connect flows now end in a real test: the owner has their own agent fire a genuine check, plainly labelled as a test, and answers it. But the backend has no test flag. A test check therefore counts against the period total while pending, and a test the owner refuses is a real refusal in denial memory. Test checks today run as ordinary asks, distinguished only by unique test items, and the product copy does not claim an exclusion the backend does not provide.

Platform trusted surfaces. AP2 verifies mandates on the platform's trusted surface; PAP's payments discussion is converging on profiles carried by the large platforms. If the eventual standard shape reserves approval to surfaces the platform itself operates, a third-party approval service has no seat, and the design in this paper survives only as an owner-side record of approvals that happened elsewhere. If, instead, the standards leave the approval evidence object open — carried by the flow, signed by whoever operated the surface — an independent surface is a feature: it is the one seat in the flow that works across agents, merchants, and rails at once. Which way this breaks is the largest external unknown for the design, and is a second reason (beyond Section 7) to put the evidence object into the standards conversation now, while the payments and signed-approvals extensions are still undesigned in public.

Owner-signed mandates. Section 5.5 conceded the point: a mandate signed with the owner's own key (the Izmaylov shape) binds the owner to the permission more strongly than a service-sealed record of an authenticated session act. The

counterweight is ceremony: key loss, device change, and recovery become owner-facing problems, and the pilot population for this design is non-technical owners. The trade is real in both directions, and this design has not taken it.

Scope. The mechanism covers purchases. PAP Operations covers any business action (exchanges, booking changes, cancellations); nothing in the evidence model is purchase-specific except the payment block, but the mandate's amount fields are, and generalization is asserted nowhere in this paper.

7. The ask: an approval-evidence object

This paper closes with a proposal to the protocols, stated for PAP Operations (where the hole is admitted in the text) and intended to travel. Define an **optional approval-evidence object** that an Operations approval (v2) or confirmation (v1) can carry, so that a company which needs more than trust can verify the approval itself. The object states:

- the operation it covers: `operation_id` and `revision` — and an operation hash, where the company computes one over the revision's terms;
- the permission it was judged under: an identifier and version of the standing permission or mandate, so "the approval was within the user's limits" is checkable rather than implied;
- who approved, and how they were authenticated: the approving principal and the authentication context (method, and freshness where the surface records it);
- the outcome — approval *or refusal* — and the decision time;
- a digest of the exact terms approved, so what was shown and what was approved cannot drift apart;
- a signature by the approval service over the canonical object, with the service's keys discoverable the way these protocols already discover things: a published key document at the service's domain.

Verification for the company is then mechanical: check the signature, check that the operation and revision match the ones it issued and have not expired, check the digest against the terms it proposed — and only then perform, under the extension's existing at-most-once rules.

Three properties, from operation, that any shape of this object should keep:

1. **Record before token.** The decision exists as a durable record before anything spendable or performable is issued from it. An attestation that can be minted before the decision is recorded attests to nothing.
2. **Refusals are first-class.** If approvals carry evidence and refusals don't, an agent (or a service) can silently retry refused actions. The outcome field covers refusal, and a company should be able to treat a recorded refusal of the same

terms as final.

3. **Approval ≠ performance.** The evidence proves the approval. What happened next — the company's `result`, or in payments the settlement — is a separate report by the performer, graded separately (Section 3), and the two are never merged into one claim.

It may be that the right carrier for this object is the AP2 / Verifiable Intent mandate profile that PAP's payments discussion is converging on. But Operations also covers non-payment actions, where no payment profile will ever be present, and §8.1's trust gap applies there in full. Two questions are therefore put to the editors: is an approval-evidence object in scope for the Operations extension itself, or only inside profiles like payments — and if it belongs in the extension, should the object be defined there, or should the extension define only the carrying point and reference a profile maintained elsewhere? The implementation behind this paper is offered as operating evidence for either answer.

References

- Personal Agent Protocol (Sierra / Meta), protocol draft 0.3 and Operations extension v2, §5, §8.1 — https://personalagentprotocol.org/docs/0.3/extensions/operations
- Personal Agent Protocol, Operations extension v1 (draft 0.2), §§5.1, 6, 8 — https://personalagentprotocol.org/docs/0.2/extensions/operations
- Personal Agent Protocol repository, issue #61 (payments extension discussion) — https://github.com/personal-agent-protocol/ personal-agent-protocol/issues/61
- Izmaylov, P., "Signed Permissions and Action Receipts for Automated Agents," draft-izmaylov-agent-permission-receipts-02, IETF, 31 October 2026 — https://datatracker.ietf.org/doc/ draft-izmaylov-agent-permission-receipts/
- Marques, J., "Compliance Profile of Signed Action Receipts for AI Agents" (Asqav), draft-marques-asqav-compliance-receipts-09, IETF (Independent Submission) — https://datatracker.ietf.org/doc/ draft-marques-asqav-compliance-receipts/
- Google, "Agent Payments Protocol (AP2)," v0.2, April 2026 (donated to the FIDO Alliance) — mandates, Checkout Receipt, Payment Receipt.
- Mastercard, "Verifiable Intent," v0.1 draft, March 2026 (pilots from February 2026); Agent Pay / Agentic Tokens documentation.
- Coinbase, "x402" protocol documentation and EIP-3009 transfer authorizations.
- Asqav product documentation (approvals, MCP security, key custody) and terms of service — https://www.asqav.com/docs
- Sealgraf production system: system description and decision records available from the author on request; the public checker is at https://sealgraf.com/check

Author

Sai Arun Gunuganti — Sealgraf. Contact: support@sealgraf.com. The author operates the system described; the operating evidence in Section 4 is the author's own deployment, with the limits stated there.