

A Payment and Entitlement Gate for Self-Serve Agentic Expert-Access

Architecture and a Formal Admission Model

King Yew Choo

True Primary

trueprimary.com

k.choo@trueprimary.com

ORCID: 0009-0002-5244-6082

Version 2.9 · 29 September 2026*

SSRN, DOI [10.2139/ssrn.7543686](https://doi.org/10.2139/ssrn.7543686). An earlier version is on SSRN as *An MPP-First, Settlement-Flexible Architecture for Self-Serve Agentic Expert-Access: Protocol Analysis, Formal Model, and Business-Model Transition* (June 2026), DOI [10.2139/ssrn.6928639](https://doi.org/10.2139/ssrn.6928639).

Abstract

A software agent buying expert work for a principal needs an answer, inside one request, to whether a purchase may start. This paper gives an architecture and a formal model for that decision. One gate state separates quotation and authority checks from fulfilment and admits a quote on one of two branches: a payment credential presented against the operator's internal payment requirement, or a plan check against an entitlement the client already holds.

Four structural facts and one invariant theorem are established over the task model: fulfilment starts only after admission; a request key yields at most one activation; a task consumes at most one admission token; a delivered artefact carries an operator-signed receipt that verifies against the operator's own records, with no independent party attesting it; and gross committed spend under a mandate stays within its cap when purchases run concurrently, under a per-mandate serialised reserve-and-commit. A corollary adds the entitlement side: a prepaid balance never goes negative under concurrent draws. A validation supplement, identified in the paper by its hash, checks the results on fulfilment, token consumption and spend, and the corollary by exhaustive search of five bounded instances of a Python encoding of the model. The Machine Payments Protocol (MPP) is the candidate wire protocol for the payment branch, and the paper fixes how an adapter binds MPP challenges to the internal requirement, what each response carries and how a refused credential leaves the task at the gate.

The model departs from the author's Internet-Draft of July 2026: it adds a separate entitlement branch and the concurrent budget result, and it differs from the draft on the timing of the payment challenge, gate refusals, deadlines and mandate revocation. Admission records an accepted credential, a billing receivable or a consumed allowance; collection of funds, provider behaviour and any deployed implementation lie outside the results.

*Working paper. An external fact is stated as of the access date its reference carries (§12).

Deployment status, 29 September 2026. This paper specifies a target architecture and a formal operating model. True Primary runs the expert-access practice described in §2, and the self-serve, machine-payable surface described here is its target end state. §4.4 tables the availability and terms Stripe states for its machine-payment methods [22, 13, 23]; True Primary’s account approval for them is not established.

Keywords: agentic payments; HTTP 402; Machine Payments Protocol; delegated authority; mandates; entitlements; expert networks; admission methods; finite-state machine; safety invariant; signed receipts.

JEL classification: D47, L86, D82, E42.

Reading guide. The general model is §§4 to 7 and 9 to 11. §2, §8 and the adoption decisions of §3 are the True Primary case and carry no evidence of their own.

Contents

1	Introduction & problem statement	4
2	True Primary: business context	9
3	Historical architecture: disciplined layering	9
4	The MPP thesis	10
5	Target architecture	15
6	Transaction model	15
7	Products & endpoints	22
8	Commercial transition: context, no evidence	24
9	A formal model of the gated fulfilment system	25
10	Safety & implementation constraints	42
11	Failure modes	45
12	Sourcing and verification methodology	48
13	Assumptions	50
14	Scope & boundary conditions	51
15	Conclusion	52
	Declarations	52
	Appendix A: Task state machine (full transition table)	53
	Appendix B: Endpoint schemas	54
	Appendix C: Protocol status and sources	61
	Appendix D: What the validation supplement must fail	62
	References	64

1 Introduction & problem statement

Expert-access and insight-services firms move value from a distributed supply of human expertise to clients who need answers under time pressure. The account-managed delivery model, as True Primary runs it (§2), puts a person at every commercial step: an account manager scopes the request, a research team finds suitable experts, a scheduler books a live engagement, and a billing function reconciles the work against a package, a subscription or an on-demand price after the fact. Each hand-off adds latency, coordination cost and judgement that is hard to standardise.

That shape has costs, and they compound.

Manual scoping and quoting. A client's question becomes a priced, bounded unit of work through conversation, and the handler sets the price; no rule computes it from the product and its parameters, so the same request can be priced differently by different handlers.

Scheduling as the default fulfilment primitive. The live expert call is the unit that practice is organised around. It needs mutual availability and a synchronous meeting even where the client needs only a bounded artefact, such as a validation or an availability check.

Delivery outside the commercial event. In that practice, work is handed over separately from the charge, drawdown or invoice behind it, and no machine-checkable record ties a delivered artefact to that event.

Billing decoupled from delivery. Payment is reconciled against an account, a package or an invoice cycle that runs apart from the work, and back-office reconciliation links the two afterwards.

The buyer this paper designs for is a software agent acting for a human principal under a bounded mandate. That is a design premise; the paper offers no adoption, volume or willingness-to-pay evidence for it. The account-managed workflow relies on human-mediated account management, scoping, scheduling and billing, and offers no uniform machine transaction contract. Published protocols cover the surrounding layers: exposing tools to models [16], delegated payment authority [34], agent-to-agent coordination [37] and commerce-channel wrappers [41, 42]. The transaction itself needs a layer in which a client presents a payment credential inside one request, challenge and retry exchange. The Machine Payments Protocol (MPP) specifies such an exchange over HTTP: a server answers a request for a paid resource with a 402 response offering the payment methods it accepts, the client retries with a credential for one of them, and the server verifies it and returns the resource with a `Payment-Receipt` header [7, 3].

This paper decomposes expert work into fixed-price, agent-callable products and places one admission gate in front of fulfilment. The architecture is **MPP-first** in product design, with MPP as the candidate wire protocol for the gate's payment branch, and **method-flexible** in implementation: the gate admits each quote under any available admission method of the branch that quote carries, payment or entitlement.

Research question. What operator-side conditions let payment, billing and prepaid-entitlement admissions share one delegated cap on gross admitted spend while an agent’s tasks run concurrently? This paper gives one sufficient construction: a per-mandate reservation taken in the same atomic step as the budget test, before admission; a single-use admission token per quote, on either branch; and an atomic decrement of the shared entitlement record the quote’s token was minted against. The formal model’s contribution is the composition of those controls over an asynchronous task lifecycle whose only edge into fulfilment is the gate, with the invariants established for it in §9.8; §1.1 sets it against the closest published work.

Contributions.

1. **An architecture** that gives each agentic protocol the layer the adoption test of §3 selects for it, over the operator’s own task engine, pricing engine, expert micro-input ledger and receipt store (§§3 to 5).
2. **A formal model of authorised admission** with two admitting branches, each with its own single-use token, a delegated mandate and a task state machine. Four structural facts are established by construction over the model and one invariant theorem is proved by induction over it, with a corollary bounding entitlement balances (§9, Appendix A); three of them are also checked exhaustively on bounded instances by the validation supplement (§12). Duties at the provider and adapter boundary are stated as obligations (§10).
3. **A wire contract** for admission over MPP: how a method adapter binds challenges to the internal requirement, what each response carries, and how refusal, escalation and the entitlement branch appear on the wire (§§6 and 7, Appendix B). **It is an application extension:** its activation departs from §11.4 of the pinned “Payment” draft [3, §11.4], and its answer to an expired challenge from the draft’s status table [3, §4.2]; §10 states both, and what a conforming deployment would do for the first.

The commercial transition of §8 is not listed among these. It is context for the design and carries no result of its own; §8 states the evidence it lacks.

1.1 Related work and positioning

The author’s Internet-Draft and adjacent drafts. The closest prior work is the author’s individual Internet-Draft of 23 July 2026 [1]. Its abstract describes a model that “places a payment or entitlement gate before fulfilment, maps that gate to adjacent protocols, and defines a task state machine”, yielding “gate-before-fulfilment safety, at most one task per retained idempotency key, at most one gate acceptance per payment requirement, and an operator-checkable record linking a delivered artefact to operator-controlled records”. Those four correspond to T1 to T4 here, and the draft “defines no protocol, wire format, or conformance requirements”. The principal differences from revision -00 are these. The draft issues one payment requirement for every quote and resolves entitlement evidence as a credential against it; §9.4 gives the entitlement branch its own predicate and its own single-use token. On concurrency, the draft takes serialisation of spend under a mandate as an operating assumption and states that bounded delegated spend is not a separate proven property; §9.7 states that serialisation as C9 and §9.8 proves the cap invariant from a reservation taken on the edge into the gate, which closes the read-then-check race an atomic commit alone leaves open. The serialisation remains an assumption of the same kind; what is new is the reservation and the invariant proof over it. On admission, §9.4 separates what admission records for external payment, billing and entitlement, where the draft routes post-gate failures to a credited-or-refunded terminal. Five further differences are rules the draft states and this paper changes or omits. The draft answers the quote itself with the 402, so a transfer can precede its authority check; here the quote carries no challenge, an unauthorised quote is answered 403, and the 402 answers the activation only once authority has resolved and the reservation stands (§6). A failed validation at the draft’s gate ends the task as cancelled, or as credited-or-refunded where a capture is recorded; here a refused activation fires no transition and the task stays at the gate (§9.4). The draft’s mandate names an intended-payee audience, and its authority predicate requires the requirement’s payee to be in that audience and the mandate to be in force and unrevoked at the current time; here the mandate carries no audience and revocation lies outside the model (§9.3). The draft tests the quoted deadline against the current time and defines deadline exits before and after its gate; here the deadline is tested only against the mandate’s horizon and no transition fires on it (§9.3). The draft counts a requirement expired from its expiry time; here a token expires only once that time has passed (§9.2). Beyond the model, the paper adds the endpoint surface, response contract and JSON of §§6 and 7 and Appendix B, the paid unlock of §9.9, the operator signature on the delivery receipt and the transition of §8. The two documents state the same currency rule for the budget test and the same holder-authentication conjunct.

The gating pattern itself is not new. *draft-schrock-action-evidence-boundary-00* requires, for consequential agent actions, “durable atomic consumption or reservation” before invocation, “closed effect outcomes, and authenticated reconciliation” [2]. The gate of §9 is an instance of that pattern for paid expert work. What this paper adds is the currency the gate admits, the two branches of §9.4, each with its own token, under one mandate and one budget invariant.

Agentic payments research. Jiang et al. formalise x402, MPP, ACP and AP2 in Tamarin against a Dolev–Yao adversary and report “missing bindings, state constraints, and cross-stage correspondences” [46]. Their lifecycle runs from intent formation through authorisation, settlement, delivery and failure recovery, and their property families include “Single Payment, Single Effect”, “Budget and Metering Conservation” under sequential and concurrent execution, and the relation that “Expensive transitions occur only after the required admission, capacity, or cost checks”; T3, T5 and T1 are the counterparts of those three at one operator’s task layer. This paper specifies that layer as an application model under an operator trust assumption, with the two admitting branches of §9.4 each consuming its own single-use token under one mandate reservation, and proves its results by construction and induction over it. The security of the four protocols is left to analyses of their kind; the binding of wire challenges to the internal requirement is obligation O7, which such an analysis would check. A security analysis of x402 lists a “duplicate-settlement race” and “allowance overdraft” among its flaw classes [47]; T3 and T5 are the operator-side counterparts, and the provider-side race stays with O2. A402 uses TEE-assisted adaptor signatures so that “payments are finalized if and only if the requested service is correctly executed and the corresponding result is delivered” [48]; the gate here gives admission-before-fulfilment ordering under an operator trust assumption. The Agentic Settlement Protocol profiles on-chain authorise-and-capture escrow for delayed fulfilment, with “atomic exposure reservation” bounding the operator’s credit risk [49]; T5 bounds the principal’s delegated spend instead, across card, invoice and entitlement as well as stablecoin. APEX adapts 402-style gating to fiat rails with “policy-governed spend control” and “idempotent settlement handling” [50], and a runtime verification framework for AP2 enforces “consume-once mandate semantics” under retries and concurrency [51]. A security analysis of AP2 finds that inputs “that shape a transaction before authorization remain outside” the protection its signed mandates give [52], the same limit this paper states for an in-bounds purchase (§11). A systematisation of blockchain agent-to-agent payments names “payment-service decoupling” among its key challenges [53]; within one operator’s services, the gate addresses it by making admission the only entry to fulfilment.

Compensation, caveats and delegated grants. Three literatures the gate instantiates directly. The reserve-then-commit-or-release pattern with a compensating record is the Saga model, which runs a long-lived transaction as a sequence of steps, each with a compensating step, in place of holding a lock [17], and its REST form, Try-Cancel/Confirm, which reserves a resource and then either confirms or cancels it [18]. The gate takes the shape while leaving the machinery: there is no distributed coordinator, the compensating record is written by the operator alone, and nothing outside the operator’s stores participates. The mandate, with budget, scope, expiry and criteria bounds, is a Macaroon in form, an authority token attenuated by caveats the verifier checks [19]; it differs in that the caveats are evaluated only by the operator against its own registry, with no carrying or re-attenuation by the holder. Delegated authorisation of a software agent acting for a principal is the subject of GNAP [20]; this paper does not use its protocol and fixes the authority question inside one gate evaluation instead.

Fair exchange and atomicity. Optimistic fair-exchange protocols avoid the trusted third party in the normal path and invoke it for exceptional recovery or resolution [54], and strong fair exchange is unattainable without a trusted third party in the general asynchronous setting [55]. NetBill withheld an information good until its server approved the transaction, releasing the decryption key in a server-signed receipt [56], and Tygar distinguishes money atomicity, goods atomicity and certified delivery [57]. Here True Primary operates the gate, its stores and the receipt, so the architecture provides operator-attested provenance and no fair exchange between mutually distrusting parties. The model prevents fulfilment from starting before admission and permits admission without completed delivery, so it establishes neither goods atomicity nor certified delivery in Tygar’s sense; the admission update itself is atomic under the operator-execution assumptions (A3, A9).

Capability-based authorisation. The mandate M is an attenuated capability: an unforgeable token (A1) conveying the authority its issuer delegated and nothing beyond it [58]. Miller, Yee and Shapiro call authority that is “exercised, but not selected, by its user” ambient, and count among the properties of capability systems that a subject selects the authority it uses for each access [59]. The mandate makes the delegated authority explicit in the request and narrows it to stated bounds, so each request selects the authority it exercises. A request outside the bounds is refused before the gate, and an incorrect purchase inside them can be admitted, its spend limited by the budget cap (§11).

Idempotency. T2 applies the discipline for at-most-once effects over at-least-once delivery, a stable client key and a unique-store compare-and-set [60], to the activation of a paid task. The HTTP Idempotency-Key draft, which expired at revision -07, specifies the same header for non-idempotent methods, its value a quoted string, answering reuse of a key with a different payload with 422 and a concurrent duplicate with 409 [4]. T3 applies a separate compare-and-set to the quote’s admission token, so the two guarantees can fail independently.

Credence goods and platform economics. Expert validation is a credence good: the buyer cannot fully verify quality even after delivery [61, 62]. The delivery receipt and the expert-criteria invariant (C4) are the machine-checkable residue of the vetting an account-managed practice performs, and neither verifies the quality of the work. True Primary’s proposed role may create cross-side effects between clients and experts, whose pricing implications depend on who controls selection, prices and the customer relationship, and on participation responses, which this paper does not estimate [63, 64].

2 True Primary: business context

True Primary (TP) provides expert-access and insight services to investment, strategy and product teams, sourcing the subject-matter experts who answer their questions under time pressure. Work is delivered into North America, Europe and Asia-Pacific from a London desk. Its clients and experts transact through conventional, non-crypto payment methods, and any forward architecture is constrained by that fact.

Three facts about that practice bear on the design. The central on-demand unit has been the full-hour expert call, which is the unit §8 converts. “Micro-input” denotes the internal decomposition of an engagement into smaller expert contributions, checks, reviews and artefacts, each an entry in the append-only ledger L of §9.6, while the client sees one fixed price. And TP is building self-serve agentic lanes with MPP as the candidate wire protocol for their machine payments, adopting protocols one layer at a time under the test of §3.

3 Historical architecture: disciplined layering

Each candidate layer was evaluated against one adoption test with three parts: whether it maps to a durable business function, whether it advances the client-to-expert workflow end to end, and whether it can become the default path for TP’s clients and experts. A layer meeting all three became core. Appendix C gives each protocol’s status with its sources; this section records the decisions and their reasons.

3.1 System of record and access

REST and TP’s internal domain services are the system of record for accounts, packages, subscriptions, pricing, task state, expert operations, billing, receipts and payouts. Agent protocols sit above that domain model and never replace it.

The Model Context Protocol (MCP) lets servers expose tools, resources and prompts to AI applications [16, 29]. Its revision 2026-07-28 made the protocol stateless, removing the initialisation handshake and protocol-level sessions, and moved Tasks out of the core protocol into an official extension, `io.modelcontextprotocol/tasks` [31]. MCP is core for the access layer only: it carries no semantics for payment, delegated spending authority, pricing or fulfilment. Over MCP each endpoint of §7 is one tool taking and returning the objects of Appendix B as tool arguments and results; the wire exchange of §6 stays on HTTP (§6.1). MCP Tasks are adopted as an interface mapping onto TP’s own task engine, whose lifecycle (authority checks, admission, expert matching, review) goes beyond a request-level state machine and whose fourteen states §9.5 fixes.

3.2 Delegated authority

The Agent Payments Protocol (AP2), announced by Google on 16 September 2025, builds trust through mandates, which the announcement describes as “tamper-proof, cryptographically-signed digital contracts that serve as verifiable proof of a user’s instructions” [34]. Its v0.2 release uses Checkout and Payment Mandates, “each existing in two stages”, open and closed [35]. TP’s asynchronous work routinely involves delegated, human-not-present decisions: a client may authorise an agent to spend up to a budget cap, within a scope, by a deadline, using only experts who meet stated criteria, with a human brought back in above a threshold. That is the mandate M of §9.3. The mandate model is core for the delegated-authority layer as one input to the gate; it records who authorised what and does not price, source or deliver.

3.3 Coordination and commerce channels

The Agent2Agent (A2A) protocol separates its scope from MCP’s in its own materials: “MCP is for agent-to-tool communication” and “A2A is for agent-to-agent communication” [37]. A2A is adopted conditionally, where TP runs a coordinating agent that talks to other agents; a client agent calling TP tools is served by MCP with fewer dependencies.

The Universal Commerce Protocol (UCP) “standardizes the full commerce journey” [41], and the Agentic Commerce Protocol (ACP), developed by Stripe and OpenAI, keeps the business as merchant of record [42]. Both are adopted conditionally, as wrappers where TP lists products on an external agentic commerce surface; TP’s task engine, pricing, mandate model and gate sit behind them. Merchant participation in UCP on Google’s surfaces is by waitlist [40], and secondary reporting from March 2026 describes Shopify merchants’ products found in ChatGPT being bought on the merchant’s own online store [43].

3.4 x402

x402 is a protocol for payments over HTTP created by Coinbase and now under the x402 Foundation [44]. Its version 2 HTTP transport signals a payment requirement with status 402 and carries the exchange in its own headers, PAYMENT-REQUIRED, PAYMENT-SIGNATURE and PAYMENT-RESPONSE [45]; MPP uses the same status code with the “Payment” HTTP authentication scheme (§4.2). TP piloted the x402 paid-resource flow in paid MCP and API experiments. x402 is not adopted as the client-facing default, because TP’s clients and experts transact through conventional payment methods; the paid-resource pattern carries forward into the gate of §4.

4 The MPP thesis

4.1 The missing transaction layer

The layer assignments of §3 leave one function without an owner: authorised admission between an agent-readable product surface and the fulfilment of expert work. The thesis places one admission gate there. The gate is the task state `pending_payment_or_plan_check`, written g^* in §9, and the fulfilment region Φ comprises the states after it (§9.5).

4.2 What MPP is

The Machine Payments Protocol (MPP), co-authored by Tempo and Stripe, is a protocol for machine-to-machine payments whose specification Tempo Labs and Stripe maintain [6]. It launched alongside Tempo Mainnet on 18 March 2026 [9, 10]. The MPP documentation lists eleven payment methods and a custom option, including Tempo, EVM, Stripe, card, Lightning and Solana [8]. Its core specification is the “Payment” HTTP authentication scheme, an individual Internet-Draft by three Tempo Labs and two Stripe authors, pinned in this paper at revision -01 of `draft-httpauth-payment` (§12) [3]; the authors of the scheme the paper adopts are affiliated with the two vendors that maintain MPP. The specification repository’s own link for its Internet-Draft points to `draft-ryan-httpauth-payment`, the earlier name [6]; this paper pins `draft-httpauth-payment-01`, and a deployment checks its adapter against the exact revision it implements, because the repository link does not establish that an adapter matches either text.

On the wire, a server answers a request for a protected resource with 402 `Payment Required` and one or more `WWW-Authenticate: Payment` challenges; the draft lets a server return several challenges in one response, “each with a different payment method or configuration”, and the client sends one credential for the challenge it selects, in `Authorization` or, where that challenge carries a header parameter, in `Payment-Authorization` [3, App. B.4]. A credential that fails validation is answered 402 with a fresh challenge, and a verified payment that policy refuses is answered 403 [3, §4.2]. On success the server returns the resource and, as the draft recommends, a `Payment-Receipt` header [3].

In the model, $R = \langle \text{amt}, \text{cur}, \text{payee}, \eta, \text{scheme}, t_{\text{exp}} \rangle$ (5) is the operator’s internal payment requirement for a quote. A method adapter builds the wire challenges from R , one per offered method in Ext , with the parameters revision -01 requires (`id`, `realm`, `method`, `intent`, `request`) and the two it recommends, `expires` and, on a request carrying a body, `digest`, and maps a verified credential for any of them to R ’s single nonce η . A billing method takes no wire challenge. The results of §9 concern R and that nonce; the binding an adapter owes is obligation O7 (§10).

The `Payment-Receipt` header is a wire payment receipt, emitted at the selected method’s own success event; for the card method that event is authorisation, “even though final fund settlement is pending” [12]. The delivery receipt Ω (18) is a separate record, signed by the operator and written once the artefact exists (§9.6).

4.3 MPP-first, method-flexible

MPP-first is a position about product architecture: MPP is the candidate protocol for the wire exchange of the payment branch, and the self-serve surface is shaped around its 402 flow, so admission completes within the activation request (§§6 and 9.5). **Method-flexible** is a position about implementation: the admission method is a layer separate from MPP’s wire exchange, and a deployment chooses which of the eight methods of $\text{Method} = \text{Ext} \uplus \text{Bill} \uplus \text{Ent}$ (9) it offers per client, jurisdiction, currency and price.

A quote fixes its branch when it is issued (§9.4). On the payment branch, card, wallet, Shared Payment Token (SPT) and stablecoin answer R with a wire credential, and invoice answers it with a billing approval and no wire challenge.

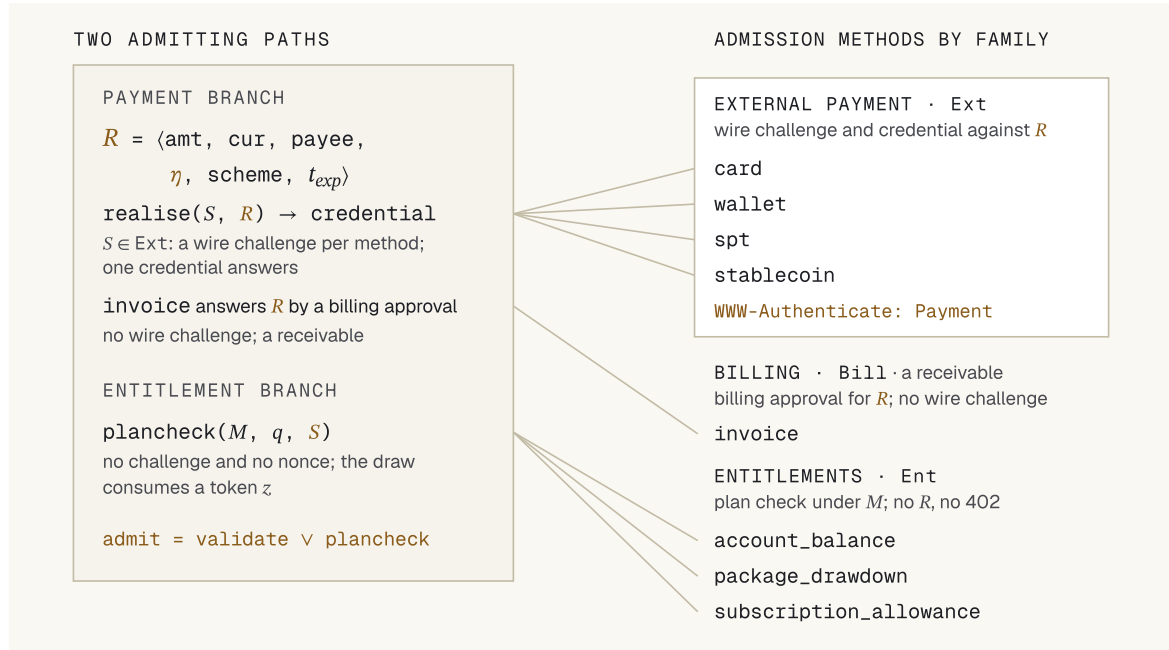


Figure 1. The eight admission methods by branch and family. The boxed group is the wire exchange: the four external-payment methods answer R with a credential, one WWW-Authenticate: Payment challenge each. Invoice answers R with a billing approval and takes no wire challenge; account balance, package drawdown and subscription allowance are admitted by plan check with no R issued. A quote fixes its branch when it is issued.

What wallet names here, and what it does not. A wallet credential is one a customer's own wallet provider presents directly for R , distinct from spt , where the customer's card or wallet is reached through a token the agent holds [13], and distinct from stablecoin , which is pre-funded on-chain. **No provider in §4.4 presents a wallet credential, so no deployment described here can offer it on the paper's date.** It is in Ext because the gate treats it exactly as it treats card : a provider outcome for a credential carrying R 's amount and currency, mapped by an adapter to R 's single nonce. A deployment that enables it names the provider and its challenge shape, and nothing in §9 changes. On the entitlement branch, account balance, package drawdown and subscription allowance are admitted by plan check, and no R is issued. The gate state and its common rules are the same for all eight, and the delivery receipt records the admitting method in its *mode* field. A method is offered for R only where its provider presents in R 's currency (§9.4).

4.4 Payment-method availability and terms

Which external payment methods a deployment can offer depends on its providers' terms. Stripe states the following for its two machine-payment methods, each fact as of the access date of its reference. Both run on preview API versions, and they are not the same version: the shared-payment-token endpoints are called with `Stripe-Version: 2026-04-22.preview` and the business-profile and crypto deposit-address endpoints with `Stripe-Version: 2026-07-29.preview` [22, 21], so both rows carry [Draft] under the tag rule of §12.

Method via Stripe	Where a business can accept it	Terms that bind the gate	Source
Shared Payment Tokens (a customer's card or wallet)	Businesses in all US states; outside the US, a listed set of 34 countries, which includes the US and GB, so 33 lie outside the US	Accepted in all Stripe-supported currencies; seller terms are the "Stripe Agentic Commerce Seller Services (Preview)" document, last modified 28 September 2026; the token carries a currency, a maximum amount and an expiry, set by the agent that issues it; API version 2026-04-22.preview [Draft]	[13, 22]
Stablecoin (MPP: USDC.e on Tempo, USDC on Solana)	Businesses in all US states except New York; outside the US, access by request in more than 30 countries. GB is not among the 33 business locations the stablecoin page lists , the US and 32 in a private preview for the EU, Hong Kong, Mexico and Switzerland, so access for a UK entity is by request	Presentment currency USD, other currencies in private preview; manual capture not supported; refunds return as stablecoins to the customer's original wallet; no dispute support; customer transaction limit 10,000 USD per transaction; API version 2026-07-29.preview [Draft]	[13, 23, 21]
True Primary account state	Not established in this paper	None	None

Two things follow for the gate, and the second is why §6 orders the exchange as it does. A GBP quote cannot be answered by Stripe's stablecoin method, whose presentment currency is USD, so that method is offered only for USD quotes. A stablecoin transfer is pre-funded: MPP on Stripe has agents "send payments to" a deposit address [21], which is why a second transfer for an admitted quote is a refund case (§11), and why §6 issues its challenges only at the gate (§6 step 5).

The operator manages no stablecoin wallet keys in this flow. Stripe provisions the deposit address through its API, off-ramps what arrives, and settles it to the Stripe balance in the business's local currency [13, 23, 21]. Under Stripe's GB terms the deposit address is a wallet address of a settlement services provider the terms list (Paxos Trust Company, LLC, or a Bridge entity), and customers deposit into the operator's own account with that provider, under Settlement Services Terms the operator accepts with it; the provider holds the stablecoins in custody and converts them [24]; the operator's interest in them before conversion is a question for Stripe's terms, the provider's own terms and O6. This removes the operator's own wallet management. **It does not remove the operator's exposure.** Stripe's published GB stablecoin payments terms, updated 16 April 2026, place the settlement service at the user's own risk, disclaim Stripe's liability for losses arising from an incorrect deposit or refund address, and disclaim liability for losses arising from a depeg event, including the difference between the rate at which a payment is accepted and the rate at which the conversion executes [24]. Stablecoin payments there support no disputes and cannot be reversed once submitted. So the custody and conversion *operations* sit with the provider, and contractual, settlement and depeg exposure remains with the operator. Whether a particular entity is eligible for the service, and on what terms, is a contract question this paper does not resolve.

The asymmetry a UK operator should read off this table. GB is on the shared-payment-token country list and is not on the stablecoin business-location list. On the paper's date a UK accepting entity can take agent card payments through SPT and cannot take stablecoin payments without requesting access, which is the one MPP method on MPP's native rail. The London desk of §2 and the stablecoin row are to be read together.

Method limits against the products of §7. Stripe states a 0.50 USD minimum for SPT card payments and a 10,000 USD customer transaction limit for stablecoin payments [13, 23]. A deployment checks each deployed price against the selected provider's current limits; this paper withholds its prices (§8) and establishes no such fit.

The table records a provider's commercial terms and no regulatory finding. Whether an operator whose customers deposit stablecoins into its own account with a settlement services provider, which converts them and settles fiat, falls within the UK cryptoasset regime is the O6 determination, and this paper answers no part of it. The regime's own dates: The Financial Services and Markets Act 2000 (Cryptoassets) Regulations 2026 were made on 4 February 2026, the FCA published final rules and guidance on 30 June 2026, and the FCA states that the regime "is expected to come into force on 25 October 2027" [27]. The GENIUS Act, law since 18 July 2025, is the United States framework for payment stablecoins [28]. Assumption A4 states the variation the model assumes.

The shared gate centralises the common admission rules while method-specific validators sit behind it. A deployment hard-coded to one method loses availability when that method is unavailable; a change in a provider's terms changes which members of *Method* a deployment offers and leaves the gate and the results of §9 unchanged.

5 Target architecture

The target architecture is a stack that keeps each protocol and each TP internal system in one layer. A request enters at the top, descends through an access layer, a decision layer and an admission-method abstraction, and crosses the gate into the fulfilment region; a delivered artefact and its signed receipt return to the caller. MPP occupies one tier: the wire protocol through which a method adapter carries R to the caller (§4.2).

The gate g^* is the single point at which the system commits fulfilment cost. Above it lie quotation and authority; below it lies Φ , none of whose states executes until $\text{admit}(q, S, M)$ holds. The gate admits on the two branches that §9.4 defines.

Product interface (access layer). An agent reaches TP through MCP or REST over the system of record. The layer routes a typed request inward and neither prices nor admits it.

Decision layer. The pricing engine evaluates $\text{price} : P \times Params \rightarrow Money$ and freezes the result at quote time. The authority component evaluates $\text{authorises}(M, q)$: authentication of the mandate holder, scope, budget headroom and deadline (§9.3). The entitlement component decides whether an account balance, a package or a subscription covers the quote; a covered quote carries a drawdown token z , minted against the one record the entitlement component selects to cover it, and any other quote carries R .

Admission-method abstraction. All eight admission methods sit behind one gate state. R is the same whichever payment method answers it; the wire challenges an adapter issues are method-specific, and each maps to R 's nonce (O7).

Fulfilment region Φ . The task engine advances an admitted request through the post-gate states of §9.5. Expert sourcing selects contributors against the criteria predicate χ carried in the mandate, and the append-only ledger L records each contribution. Validation, review and aggregation produce the artefact; $F : ValidatedReq \rightarrow Artefact$ is partial because an admitted task can end without one (§9.6).

Receipt store. At delivery the operator writes and signs the delivery receipt Ω (§9.6), which a verifier checks against the operator's records. Expert payout runs through TP's payout or invoice process outside this client-facing path.

6 Transaction model

The machine-payment transaction is an eight-step exchange between a client agent and TP, gated on one admission decision. On the payment branch it follows the MPP challenge, credential and receipt pattern [7, 3]: the 402 answers the activation request unless it selects invoice (step 3), and the credential returns on that request's own body.

1. **Request.** The agent calls a product endpoint (§7) over MCP or REST, names its mandate by registry identifier, and authenticates as that mandate's holder, by HTTP Message Signatures [5] or mutual TLS. The draft asks a server to verify authentication before issuing a payment challenge [3, §4.4.3]. TP enters a task under the mandate at `draft`.

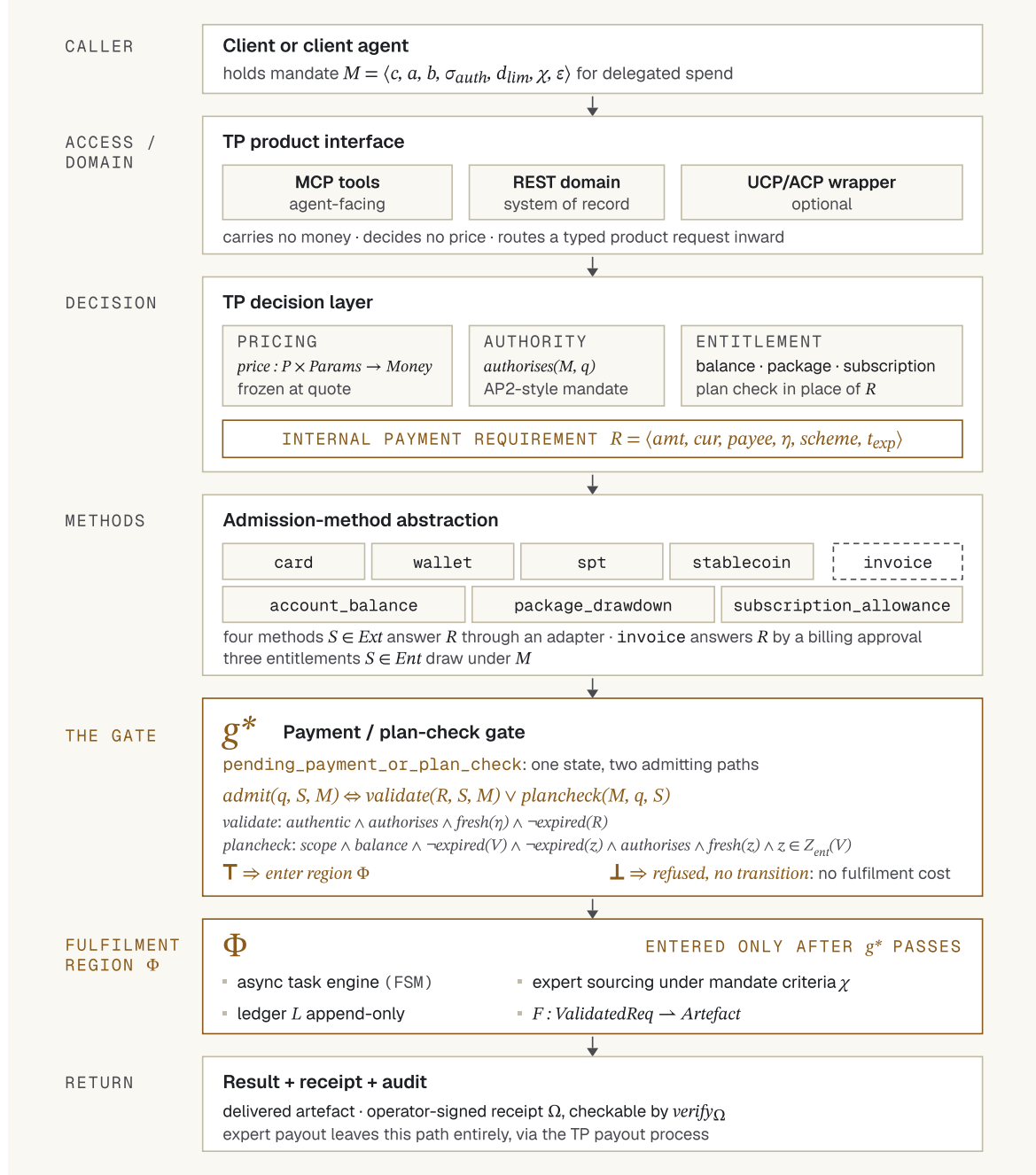


Figure 2. The reference stack from caller to receipt. The decision layer turns a request into a quote carrying either the internal payment requirement R or a drawdown token z ; a method adapter builds MPP challenges from R ; one gate state g^* admits on either branch before the fulfilment region Φ .

2. **Quote.** The pricing engine freezes the price, and TP evaluates $\text{authorises}(M, q)$ on the frozen quote before issuing anything. **A quote the mandate does not authorise is answered 403 with no token and no challenge**, so no credential is ever obtained for it and no pre-funded transfer is invited; the task does not leave *draft*. Otherwise TP issues the quote with its admission token, moving the task to *quoted*, and the quote response carries no challenge on either branch: on the payment branch it is 200 with the quote, the fields of R printed in Appendix B.1 and the offered methods; on the entitlement branch it is 200 with the quote and its drawdown token z . Where the mandate's escalation predicate holds for the frozen quote, the quote is issued with 202 and a status link, and no challenge is issued for it until the principal's decision is recorded (step 4). This reads the predicate to shape the response; the transition it guards is taken at *pending_authority* (§9.5).
3. **Choose the method, where the quote offers more than one.** A quote may offer a method in *Ext* and *invoice* together, as the worked quote of Appendix B.1 does. *Invoice* is selected by "admission_method": "invoice" in the body of the activation request (step 4); an external method is selected by presenting its credential on the retry (step 6). Where the quote offers *invoice* alone, its activation request selects *invoice* without a selector. A first arrival that does not select *invoice* draws the 402 of step 4. The selector names a method the quote already offered and changes no guard of *admit*. On the entitlement branch the request draws on the record its drawdown token names; a body whose "admission_method" names another of the client's entitlements draws on that record and is refused (§9.4).
4. **Activate.** The agent sends the activation request: the quote identifier, the request parameters, the mandate identifier and an *Idempotency-Key*, authenticated as the holder and carrying *Content-Digest* over its body. TP processes it in the order of §9.5: it checks the parameter digest against the quote, binds the key, takes the task through *accept* and *begin authority check*, resolves authority and escalation at *pending_authority*, and takes the edge into the gate, reserving the price against the mandate. On the payment branch this first arrival carries no credential, and the method it selects (step 3) decides the answer. **Where it selects invoice**, whether the quote offers *invoice* alone or beside methods in *Ext*, no wire challenge is issued: a recorded billing approval for R admits the request on this first arrival with 200, and an approval still pending is answered 202 with a status link, the task waiting at the gate. **Where it does not select invoice**, it presents no admission method, satisfies neither disjunct of *admit* and is refused at the gate with no transition (§9.4); TP answers it 402 with one *WWW-Authenticate: Payment* challenge per method in *Ext* that the quote offers, each built from R and each carrying as its *digest* parameter the digest TP computes over the body that drew it [3, §5.1.3], and the task stays at the gate with its token fresh and its reservation standing. No such 402 lacks a challenge, which the draft forbids [3, §4.4.2]: a quote with no method in *Ext* offers *invoice* alone, and its request selects *invoice*. On the entitlement branch the request carries no credential at any point and runs on to step 6. Where escalation holds and no decision is recorded, the request is answered 202 with a status link and the task waits at *pending_authority*; a later arrival of the same request, once the decision is recorded, takes the edge into the gate and is answered as this step states.

5. **Pay (payment branch).** The agent selects one offered method and obtains a credential for its challenge. For a pre-funded method, such as stablecoin, the transfer happens at this step, after authority has resolved and the reservation stands, which is why neither step 2 nor step 4 issues a challenge for a quote the mandate does not authorise. A mandate that expires *after* step 4 is caught at step 6, and a transfer already made is then an unpaired provider effect, paired for a refund under O4. **Exhaustion is not caught there for this quote:** the reservation taken on the edge into the gate (C9) already holds this quote's amount, so the budget conjunct it discharges cannot fail at the gate on account of a later activation under the same mandate. That is the point of the reservation, and it is why a concurrent activation under the same mandate, finding no headroom, is refused on the edge into its own gate, at `pending_authority`, and goes to `cancelled` (Appendix A). A mandate revoked after step 4 is not caught there, because A1 assumes mandate status holds for its lifetime and authorises reads no status (§9.3, §11).
6. **Admit.** The agent retries the activation request with its body unchanged, byte for byte, under the same Idempotency-Key, adding one credential in `Authorization: Payment`; the challenge's digest parameter binds the credential to that body, and a server rejects a credential presented on any other [3, §5.1.3]. Within that request TP checks the parameter digest again, proceeds against the task the key is bound to, and evaluates the gate in the order of §9.5: the local conjuncts, among them authorises re-asserted and the standing reservation, then the provider's outcome for the credential. When admission holds, one atomic update consumes the token, commits the reservation, writes the task record and moves the task to `pending_expert_match`.
7. **Respond.** Only after that update does TP return 200 with the task identifier, state and admission disposition (Appendix B.3). Any `Payment-Receipt` header, the selected method's wire payment receipt (§4.2), travels on this response, marked `Cache-Control: private`.
8. **Deliver.** Delivery is a later, separate event. At the `ready` → `delivered` transition TP writes and signs the delivery receipt Ω , and the artefact and receipt are retrieved through the contracts of §7.

Figure 3 shows the payment branch with header names as they appear on the wire. The activation request is answered as follows.

Condition	Response	Task afterwards
Client authentication fails	401	unchanged
Parameter digest differs from the quote's, or the key is already bound to a request with another digest, another quote or another mandate [4], or the quote's task already carries another key (C8)	422	unchanged; the request binds no key
A replay: from the holder of the bound task's mandate, presenting that mandate, under a key bound to this quote and digest, after the activation has resolved	the recorded outcome (410 with the task state where the activation ended without admission) and, after delivery, the receipt; no second admission and no provider call	unchanged
An overlapping invocation under the same key is still executing [4]	409	unchanged
Escalation holds and no approval is recorded	202 with a status link	pending_authority
The mandate stopped authorising the quote after it was issued	403	cancelled from pending_authority; at the gate, unchanged
No credential, invoice not selected (step 3), and the quote offers a method in <i>Ext</i>	402 with fresh challenges	at the gate; reservation standing
No credential, invoice selected (step 3), alone or beside methods in <i>Ext</i> , and a billing approval for <i>R</i> is recorded	200	pending_expert_match
No credential, invoice selected (step 3), and no approval is recorded	202 with a status link	at the gate; reservation standing
Before admission: credential malformed, challenge unknown or used, or verification refused by the provider	402 with fresh challenges	stays at the gate
Provider outcome indeterminate	202 with a status link	stays at the gate (O3)
The provider requires customer action	202 with a status link and the provider's action reference	stays at the gate (O3)
Entitlement branch: the record drawn on does not hold the quote's drawdown token, does not cover its scope, or holds less than the quoted price	403 with the task state; no challenge	at the gate; token fresh, reservation standing
Admitted	200	pending_expert_match
Token or entitlement expired, or the task has ended, on a request that is not a replay	410 with the task state	cancelled or its terminal state

The reservation stands from the first arrival of the activation request, so a challenge is issued only for a quote whose headroom is already held. A refused credential therefore costs the agent one round trip: the quote, the reservation and the other offered methods stay available until the token expires (§9.4), and a client that draws challenges and then decides against paying withdraws at the gate and gets its headroom back at once (§7). Offering several methods in one 402 lets a client choose without a second request. Every 403 carries no challenge and names its cause in a problem body of B.3's shape, its type in the operator's own namespace: `mandate-not-authorising` where the mandate does not authorise the quote, and, on the entitlement branch, `entitlement-insufficient` for a balance below the price or `entitlement-mismatch` for a record that does not hold the token or cover the scope. After an entitlement refusal the client sends the request again under the same Idempotency-Key, drawing on the record the token names, once a top-up or renewal outside the model makes that record's balance cover the price, or withdraws at the gate, which releases its reservation, and requests a new quote (§9.4). A client that funds two pre-funded methods for one quote has the second transfer paired for a refund under O4 (§11).

6.1 MCP binding

§3.1 makes MCP the access layer and §4.2 makes MPP the wire protocol for the payment branch, and the two meet somewhere. An MCP `tools/call` is a JSON-RPC request whose result is a JSON object; the exchange above is specified in HTTP terms, a 402 carrying `WWW-Authenticate: Payment`, a retry carrying `Authorization: Payment` and a `Payment-Receipt` on the response. Two published patterns join them and they differ in what travels on the wire, so the paper picks one.

This paper binds MCP by the link pattern. The MCP tool takes the objects of Appendix B as its arguments, validates the request and *returns the quote object and an activation URL as tool-result data*. The 402, the credential and the `Payment-Receipt` travel on the REST activation endpoint of §7. No 402 is ever returned on `tools/call`, and the `Payment` scheme's headers are never carried inside a JSON-RPC result. This matches the pattern Stripe publishes for monetising an MCP server, in which the tool “returns a payment link” pointing to “a separate HTTP endpoint that handles MPP” [14]; Stripe labels that guide experimental.

The alternative, and why the gate does not care. A client whose MCP transport wraps `fetch` can receive the 402 on `tools/call` itself: MPP's own MCP SDK installs a payment-aware `fetch` that the Streamable HTTP transport uses, which on an MPP challenge “creates a Credential, retries the call, and returns the result” [15]. Revision 2026-07-28 of the MCP specification removed sessions and the initialize handshake, which makes a per-request 402 on the transport feasible [16], but whether a given client surfaces it or raises a transport error is a property of that client, and the protocol settles it neither way. **The gate is identical under both.** §9.5's six steps run inside the activation request either way; only the layer the challenge travels on differs, and O7 binds the adapter's challenges to the one nonce in both cases. A deployment that offers the transport pattern states so in its tool description, and the paper's results are indifferent to the choice.

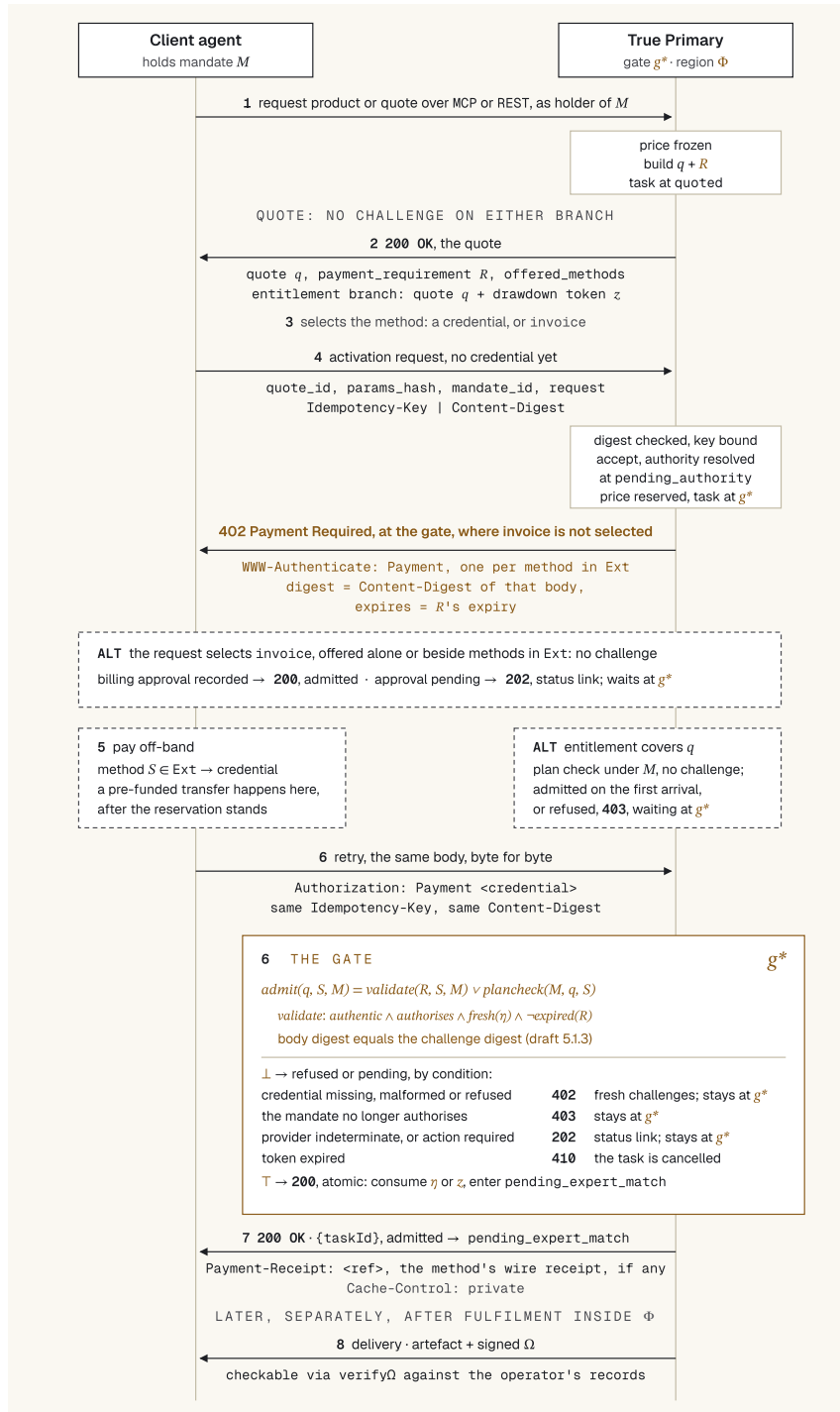


Figure 3. The exchange on the payment branch, numbered as the steps of §6: the quote, with no challenge; the activation request, which, unless it selects invoice, draws the 402 at the gate with each challenge’s digest bound to the body that drew it; the credential on a retry of that body. An activation that selects invoice, alone or beside external methods, takes no challenge and is answered 200 where a billing approval is recorded and 202 while one is pending. The gate answers the retry by condition: 402 with fresh challenges for a missing, malformed or refused credential, 403 where the mandate no longer authorises, 202 with a status link while the provider outcome is indeterminate or customer action is required, and 410 on an expired token; admission returns 200 with any *Payment-Receipt*. On the entitlement branch the plan check admits on the first arrival or refuses with 403 and no challenge. Delivery with the signed receipt Ω is a later, separate event.

A worked exchange (target state). A client's research agent holds a mandate capping spend at £1,000 within an authorised scope and calls `POST /self-serve/fixed-price-brief` for a single-question brief. TP freezes an illustrative price of £500 and returns the quote with its offered methods. The agent sends the activation request; TP checks the digest, binds the key, finds the escalation predicate false and the mandate authorising the quote (holder authenticated; scope; £0 standing plus £500 within the cap; deadline), reserves £500 and answers 402 with a challenge for each offered method in *Ext*, each bound to that request's body. The agent retries the same body with an SPT credential; TP confirms the credential with the provider and admits. Appendix B gives the objects field by field, with the entitlement branch in B.5.

7 Products & endpoints

The first machine-payable surface is the target product set below, in the sense of the deployment-status note at the front. Every endpoint is agent-callable over MCP or REST; every product endpoint is fixed-price, with the amount frozen at quote time, and the three reads, the artefact retrieval and the withdrawal take no payment. Three contracts are common to all of them.

- **Admission.** A quote on the payment branch carries *R*; the caller's activation request draws 402 with the adapter's challenges at the gate and is retried, on the same body, with one credential, unless it selects invoice, which draws no challenge and is answered 200 or 202 (§6). A quote covered by an entitlement carries a drawdown token and is admitted by plan check with no challenge. Both branches pass the same gate (§9.4).
- **Idempotency key.** Every state-changing call carries a client-supplied `Idempotency-Key`, its value a quoted string, as revision -07 of the expired `Idempotency-Key` draft requires [4]. A repeated quote request under the same key returns the recorded quote and creates no second one. Once the activation has resolved, a repeat under the same key, naming the same quote with the same parameters, returns the original outcome with no second admission or fulfilment (T2); before it resolves, a repeat is evaluated at the task's current state, which is how the credential returns on the request that drew its challenge (§6). Reuse with other parameters, another quote or another mandate is answered 422. The credential retry carries the key and the body of the request that drew the challenge, so a client library that issues a fresh key on its automatic retry cannot complete admission: the new key is refused (C8) and the task waits at the gate until the client withdraws or the token expires.
- **Quote and activation share one path, and `quote_id` is the discriminator.** A deployment publishes disjoint schemas for the two operations, or an explicit operation discriminator, and rejects a body carrying fields of the other; 409 applies to an overlapping HTTP invocation of the same operation under one key, never to the whole asynchronous task, so a later credential retry or status read is answered against the recorded task state (§9.5). A `POST` to a product endpoint whose body carries no `quote_id` is a quote request and is answered 200 with the quote, or 202 with a status link where the escalation predicate holds (§6 step 2): with the offered methods on the payment branch, with the drawdown token on the entitlement branch, and with no challenge on either. A `POST` whose body carries a `quote_id` is an activation and runs the six steps of §9.5; on the payment branch its first arrival carries no credential and, unless it selects invoice (§6 step 3), draws the 402 at the gate. No header separates the two calls on either branch, so `quote_id` is the only discriminator, and it is normative.

Verb	Path	Purpose	Admitted as
POST	/self-serve/expert-availability-check	Whether experts meeting the mandate's expert criteria are available within a deadline	paid task
POST	/self-serve/quick-validation	Single-pass expert validation of a supplied claim, figure or artefact	paid task
POST	/self-serve/fixed-price-brief	Async expert brief on a defined question	paid task
POST	/self-serve/market-pulse	Fixed-scope market-pulse pack of expert micro-inputs	paid task
POST	/self-serve/add-on-micro-validation	Follow-up validation attached to an existing task or artefact	paid task
POST	/paid-artifacts/{artefactId}/unlock	Unlock an existing stored artefact, such as a transcript summary	paid unlock, an extension (\$9.9)
GET	/paid-artifacts/{artefactId}	Retrieve an artefact already covered by an unlock grant or by the task that produced it. The path takes an artefact identifier, never a task identifier	retrieval, no payment
GET	/quotes/{quoteId}	Status before admission, addressed by quote identifier: task state, any recorded escalation decision, activation record and provider disposition	read, no payment
POST	/quotes/{quoteId}/withdraw	Withdraw the quote's task at the gate or at needs_client_clarification; refused elsewhere without change of task state	cancellation, no payment
GET	/tasks/{taskId}	Task status: state, activation record, admission and compensation disposition, and, once delivered, the artefact identifier	read, no payment
GET	/tasks/{taskId}/receipt	The canonical signed delivery receipt and its signature, to the mandate holder of record	read, no payment

The endpoints carry eight self-serve products: the five task products, each on the endpoint of the same name; result-artefact unlock and transcript summary, both sold through the unlock endpoint; and the API insight call, the calling convention all of them share.

Lifecycle contracts. Each is a read of, or an instruction naming, machinery §9 defines. *Replay*: a reissued call follows the idempotency contract above; once the activation has resolved the agent observes the same activation record $\text{act}(k)$ and, after delivery, the same receipt (T2). *Status*: before admission, a read addressed by quote identifier returns the task state, any recorded escalation decision, any activation record and any stored provider disposition; after admission, it returns $\text{st}(\text{taskId})$ with the admission and compensation disposition read off the task record, and for a delivered task the artefact identifier that `GET /paid-artifacts/{artefactId}` takes. An identifier that is undefined, or whose caller is not the mandate holder of record, is answered not-found. *Clarification*: an answer leaving the quoted parameters unchanged resumes fulfilment at the frozen price (C1); an answer changing them needs a new quote and a new task, since `REQUOTE` is enabled only at quoted. *Cancellation*: a client may abandon at draft, reject at quoted, withdraw at the gate, or withdraw at `needs_client_clarification`; the last ends at `compensation_recorded`. Abandon and reject are client events of the model with no endpoint of their own here, and a decline at `pending_authority` is the recorded escalation decision (§9.3). Withdrawal at the gate takes the task to `cancelled` and releases its reservation, so a client whose provider declined, or which decides not to pay, recovers its own mandate headroom without waiting for the token to expire. It consumes no token, so T1, T3 and T5 are unaffected: the edge leaves $Q \setminus \Phi$, nothing is consumed, and a release only lowers $\text{tot}(M)$. Elsewhere the withdraw endpoint refuses the call without change of task state. *Dispute*: the model types the upheld outcome only, the edge from delivered to `compensation_recorded`; adjudication and its window are operator process, distinct from a provider-side dispute (§14). *Receipt retrieval*: returns the canonical signed receipt $\text{rcpt}(\text{taskId})$ to the mandate holder of record.

Appendix A prints time-driven exits at four states: the quote, the authority check, the gate and the clarification wait. Nothing in the model forces a task stalled in any other state to a terminal, and a deployment closes that gap with the runtime monitors of its operational layer (§14).

8 Commercial transition: context, no evidence

The transition reshapes each commercial primitive of the established model into a self-serve, agent-addressable form: the unit moves from a human-mediated engagement billed after the fact to a fixed-price product whose quote, authority, admission, fulfilment and receipt are machine-readable. The conversion acts on the unit and does not reprice the existing one, and it carries no demand, margin or conversion-rate evidence.

The change of unit spreads one expert capability across separately priced products. Absolute prices are withheld, so this section states no price relation a reader could test, and whether each deployed price sits inside the selected provider's current transaction limits is checked per deployment (§4.4).

From negotiated quote to deterministic quote. Account-managed scoping gives way to a quote computed from the product and its parameters and frozen at issue: $\text{price} : P \times \text{Params} \rightarrow \text{Money}$, surfaced as the quote q of §9.2. Paired with a mandate (§9.3), the price no longer depends on which account manager handles the request.

From scheduling by default to artefact by default. The default unit becomes a fixed-price async artefact (a brief, a validation, a market-pulse pack, an availability check) delivered with no meeting. The live call stays available as a priced bespoke exception where the work cannot be expressed as a bounded product.

From back-office billing to a transaction-bound gate. Where a machine method is practical, the gate replaces manual billing at the point of purchase: fulfilment starts only after authorised admission, and every delivered task yields a signed receipt a verifier checks against the operator's records. Where no external method is available to a client or in a jurisdiction, a payment-branch quote can offer invoice, and a client holding an entitlement is quoted on the entitlement branch.

Implemented as specified, the commercial path runs from agent discovery to a delivered artefact and its receipt without an account manager in it (§§5 to 7). Human mediation moves into the exceptions around the transaction, where the account-managed engagement is kept for the work that still needs it.

9 A formal model of the gated fulfilment system

This section models the architecture of §§5 to 7 over the paid task path, adds the paid unlock of a stored artefact as an extension in §9.9, and establishes the core safety and integrity properties by construction. The model fixes what must hold at each step and leaves the mechanism to the implementer, so a deployment on different infrastructure can be checked against the same conditions. Notation is set-theoretic with a labelled transition system for task state: $\langle \dots \rangle$ for tuples, $\rightarrow$ for partial maps, $\rightarrow$ for functions and transitions (disambiguated by context).

Sym.	Name	Defined	Sym.	Name	Defined
M	delegated mandate	(3)	q	quote	(2)
R	internal payment requirement	(5)	ω	admission token, $\omega = \text{tok}(q) \in \text{Req} \uplus \text{Draw}$	(2)
S	admission method, $S \in \text{Method}$	(9)	V	entitlement record	(11)
η	requirement nonce (single-use)	(5)	z	drawdown token (single-use)	§9.4
Ω, σ_Ω	delivery receipt and its signature	(18)	G	unlock grant	(20)
χ, ε	expert-criteria and escalation predicates	(3)	$\text{act}(k)$	activation record	§9.5
Φ	fulfilment region	(16)	g^*	gate state pending_payment_or_plan_check	(15)
L	append-only expert micro-input ledger	§9.6	F	fulfilment map (partial)	§9.6

Alongside them run the numbered families. **A1 to A9** are the assumptions of §13: mandate and credential authenticity; deterministic pricing; gate ordering is operator-enforceable; method availability varies; sufficient expert supply; hash and signature integrity; stable client idempotency keys; receipt and ledger integrity; single-use token integrity. **C1 to C11** are the well-formedness constraints of §9.7: price-freeze; scope; deadline; expert criteria; authority and budget; escalation; single requirement; idempotency-key uniqueness; serialised mandate spend; single drawdown token; serialised entitlement draw. **T1 to T4** tag the structural facts and **T5** the theorem of §9.8: gate-before-fulfilment safety; idempotency under a request key; single admission-token consumption; auditability; budget-cap soundness under concurrent delegated spend.

9.1 Principals and catalogue

C is the set of clients (principals holding accounts, budgets and authority), A the set of agents acting for clients, E the set of experts, and P the set of fixed-price products. A product is requested with parameters from Params , and prices are integer minor units with a currency tag:

$$\text{Money} = \mathbb{N} \times \text{Cur}, \quad \text{cur}(\text{amt}, c) = c.$$

$\leq$ and $+$ on Money are defined for equal currency by the arithmetic of $\mathbb{N}$. **A guard that compares or adds two amounts of different currencies is false**, so a cross-currency comparison fails closed; a quote fixes one currency, and nothing in the model converts.

9.2 Decision maps: pricing, scope, deadline

The decision layer is three total maps and a quote constructor. A request is made by an agent, and the quote records that agent so authority can be checked against it:

$$\begin{aligned}
 \text{price} &: P \times \text{Params} \rightarrow \text{Money} && \text{the pricing engine (total, deterministic)} \\
 \text{scope} &: P \times \text{Params} \rightarrow \Sigma && \Sigma = \text{scope descriptors} \\
 \text{deadline} &: P \times \text{Params} \rightarrow D && D = \text{instants, linearly ordered}
 \end{aligned} \tag{1}$$

$$q = \langle a, p, \rho, m, \sigma, d, \omega \rangle \tag{2}$$

with accessors $\text{agent}(q) = a$, $\text{prod}(q) = p$, $\text{par}(q) = \rho$, $\text{m}(q) = \text{price}(p, \rho)$, $\text{scp}(q) = \text{scope}(p, \rho)$, $\text{dl}(q) = \text{deadline}(p, \rho)$ and $\text{tok}(q) = \omega$, the admission token. The seventh field is a payment requirement R where a fresh machine payment is needed and a drawdown token z where an entitlement covers the quote, so $\omega \in \text{Req} \uplus \text{Draw}$. Every token carries an expiry $t_{\text{exp}}(\omega) \in D$, and $\text{expired}(\omega) \Leftrightarrow \text{now} > t_{\text{exp}}(\omega)$. Write $\text{ph}(q) = H(\text{canon}(\text{par}(q)))$ for the canonical parameter digest, with canon an injective serialisation of Params and H the hash of A6. The scope inputs include the question a request states, so $\text{ph}(q)$ covers the work order.

Determinism of price and the freeze constraint C1 fix $\text{m}(q)$ when the quote is issued, so an agent authorising a quote authorises the amount it will be charged.

9.3 Delegated authority: the mandate

Delegated, human-not-present authority is a mandate

$$M = \langle c, a, b, \sigma_{\text{auth}}, d_{\text{lim}}, \chi, \varepsilon \rangle \in \text{Mandate} \quad (3)$$

with issuer $\text{issuer}(M) = c \in C$, holder $\text{holder}(M) = a \in A$, budget cap $\text{cap}(M) = b \in \text{Money}$ (cumulative, one currency), authorised scope set $\sigma_{\text{auth}}(M) \subseteq \Sigma$, authority expiry $d_{\text{lim}}(M) \in D$, expert-criteria predicate $\chi_M : E \rightarrow \{\top, \perp\}$ and escalation predicate $\varepsilon_M : \text{Quote} \rightarrow \{\top, \perp\}$.

Committed spend is $\text{spent} : \text{Mandate} \rightarrow \text{Money}$. Write $\text{Cmt}(M)$ for the quotes committed under M and $\text{Rsv}(M)$ for those whose reservation against M stands neither committed nor released; both are indexed sums over quotes:

$$\text{spent}(M) = \sum_{q \in \text{Cmt}(M)} \text{m}(q), \quad \text{res}(M) = \sum_{q \in \text{Rsv}(M)} \text{m}(q), \quad \text{tot}(M) = \text{spent}(M) + \text{res}(M),$$

$$\text{tot}_q(M) = \text{spent}(M) + \sum_{q' \in \text{Rsv}(M) \setminus \{q\}} \text{m}(q').$$

$\text{tot}_q(M)$ is the spend standing against M apart from any reservation of q itself, so a quote's own standing reservation is counted once. Quotes are distinguished by their unique admission tokens (A9), which keeps two activations of the same price distinct in the sums.

Authority and escalation are two separate predicates. Both are read on the frozen quote at issue, to shape the quote response (§6 step 2), and both are resolved as transitions at `pending_authority`; `authorises` is re-asserted a third time at the gate, because a mandate can be exhausted or expire between the quote and the activation. **Re-asserting** `authorises` **does not implement revocation**: no conjunct of (4) reads a current mandate status, and A1 assumes a registered mandate remains valid until its expiry. A deployment that needs revocation to stop a later admission adds a current-status conjunct, names the authoritative registry and its version, defines the commit as atomic against it, replaces A1, and re-establishes the results that rest on it. The author's Internet-Draft carries such a conjunct; this paper does not, so that no result here depends on a status that changes outside the model. Authority alone is:

$$\begin{aligned} \text{authorises}(M, q) \Leftrightarrow & \text{authn}(\text{holder}(M)) \wedge \text{holder}(M) = \text{agent}(q) \wedge \text{scp}(q) \in \sigma_{\text{auth}}(M) \wedge \\ & \text{cur}(\text{m}(q)) = \text{cur}(\text{cap}(M)) \wedge \text{tot}_q(M) + \text{m}(q) \leq \text{cap}(M) \wedge \text{dl}(q) \leq d_{\text{lim}}(M) \wedge \text{now} \leq d_{\text{lim}}(M) \end{aligned} \quad (4)$$

where `authn(x)` holds when the activation request under evaluation is authenticated as x (A1). A mandate is named by identifier and read from the registry; its signature is verified at registration and never again at activation (A1, Appendix B.2). A quote in another currency than the cap fails the fourth conjunct; in the author's Internet-Draft it fails the budget test [1].

`dl(q)` is a **requested deadline and carries no lifecycle guarantee here**. It is tested only against the mandate's own authority horizon $d_{\text{lim}}(M)$, never against `now`, so a quote whose requested deadline has already passed can still satisfy `authorises`; and no transition of §9.5 fires on it after the gate, so an admitted task may remain in Φ past it. The model states no liveness property (§7), so this is a term the deployment gives meaning to: a deployment that intends a service deadline adds a pre-gate `now < dl(q)` guard and a post-gate expiry and remedy policy, including its races with readiness and delivery, and tests the boundary instant.

When $\varepsilon_M(q) = \top$ the task needs a recorded human decision $\langle M, q, \text{approver}, t, \text{decision} \rangle$ before it may reach the gate, and is never advanced automatically (C6); `approved(M, q)` holds when such a record with decision `approve` exists, and approval waives no conjunct of `authorises`. The expert-criteria predicate is enforced as a ledger invariant (C4), so its failure surfaces inside the fulfilment region as the no-qualifying-expert transition. **A client's pre-clearance rule belongs in ε_M and never in χ_M** . ε_M is a predicate over the whole quote and is read before the gate, and `ph(q)` covers the question a request states (§9.2), so a rule on topic, company or a restricted list can be expressed there and returns a refusal or a human decision before any charge. A rule expressed only through χ_M is enforced after admission and ends in compensation instead of refusal, because χ_M binds the expert and the ledger, leaving the quote outside it. The example registry entry in Appendix B.2 shows both limbs, an amount threshold and a restricted-list term; an amount threshold alone escalates on price where a compliance programme escalates on risk. M carries no product and no admission method: an agent holding it may buy anything whose scope lies in $\sigma_{\text{auth}}(M)$ until the budget or the expiry stops it or ε_M returns it to a human. Whether a purchase inside those bounds is correct or useful is not a condition of `authorises`.

The mandate fixes the client of record, $\text{issuer}(M)$, which Ω records. On the entitlement branch it also fixes whose records are drawn against: the request's method selects $V = \text{ent}(S, \text{issuer}(M))$, and (13) admits the draw only where the quote's token was minted against V . On the payment branch the funding instrument is whatever the presented credential draws on, and the model does not constrain it to $\text{issuer}(M)$ (O1 names the policy).

9.4 Payment requirement, admission methods and entitlements

The payment requirement is the operator's internal requirement for a quote that needs a fresh machine payment:

$$R = \langle \text{amt}, \text{cur}, \text{payee}, \eta, \text{scheme}, t_{\text{exp}} \rangle \in \text{Req} \quad (5)$$

with $(\text{amt}, \text{cur}) = m(q)$, payee the operator, η a globally unique server-minted nonce (A9), scheme the wire authentication scheme (Payment for MPP) and t_{exp} its expiry. The map req issues exactly one R per such quote, and $q(R)$ is the unique quote whose requirement is R . R is an application object: a method adapter builds the wire challenges from it and maps a verified credential for any of them to $\eta(R)$ (O7).

The gate admits on two branches. The methods in *Chal* answer R , those in *Ext* with a wire credential and invoice with a billing approval, and the entitlements draw on a balance the client already holds:

$$\text{Chal} = \{ \text{card}, \text{wallet}, \text{invoice}, \text{spt}, \text{stablecoin} \} \quad (6)$$

$$\text{realise} : \text{Chal} \times \text{Req} \rightarrow \text{Credential} \quad \text{the credential method } S \text{ produces for } R \quad (7)$$

For $S = \text{invoice}$ the credential is the recorded billing approval.

$$\text{Ent} = \{ \text{account_balance}, \text{package_drawdown}, \text{subscription_allowance} \}, \quad \text{Chal} \cap \text{Ent} = \emptyset \quad (8)$$

$$\text{Ext} = \{ \text{card}, \text{wallet}, \text{spt}, \text{stablecoin} \}, \quad \text{Bill} = \{ \text{invoice} \}, \quad \text{Chal} = \text{Ext} \uplus \text{Bill}, \quad \text{Method} = \text{Ext} \uplus \text{Bill} \uplus \text{Ent} \quad (9)$$

the eight admission methods in three economic families. **External payment** (*Ext*): admission records an accepted credential drawn on funds held outside True Primary; whether and when the provider moves those funds is the provider's own event (§11). **Billing** (*Bill*): admission records a billing approval and raises a receivable, which is paid, part-paid or written off outside the task state machine, possibly never. A billing approval is the deployment's credit decision. The model gives no bound on a client's aggregate credit exposure: T5 bounds the principal's authorised spend under one mandate, and a client with several mandates can hold approved invoices under each; a deployment that claims a client-wide credit bound supplies the control O1 names. **Entitlement** (*Ent*): admission decrements $\text{bal}(V)$ in the same atomic step, so *Ent* is the only family whose economic event the model carries to completion, and Corollary 1 is what bounds the balance it draws on. Admission under any of the eight establishes that admit held at g^* . It establishes nothing about whether funds have moved outside True Primary, and the model establishes neither cash collection nor refund completion. No guard after the gate tests a receivable, a settlement or a provider-reported outcome, so a task admitted on *Bill* runs to delivered whether or not its invoice is ever paid.

Authenticity by family. For $S \in \text{Ext}$, $\text{authentic}(\text{cred}, R)$ holds when the method's provider has returned a definitive success for a credential carrying exactly R 's amount and currency, requested under the provider idempotency key of that provider attempt (O2); for a pre-funded method the success is the provider's confirmation of the transfer. For $S \in \text{Bill}$ it holds when a billing approval for R is recorded within the client's credit limit. A method whose provider cannot present in $\text{cur}(R)$ produces no authentic credential for R and is not offered for it (O1).

With $\text{fresh}(\eta)$ meaning η has been issued and neither consumed nor withdrawn, validation is:

$$\text{validate}(R, S, M) \Leftrightarrow \text{authentic}(\text{cred}, R) \wedge \text{authorises}(M, q(R)) \wedge \text{fresh}(\eta(R)) \wedge \neg \text{expired}(R) \quad (10)$$

Here cred is the credential the activation request presents under S , and M is always $\text{md}(r)$, the mandate bound to the task when TASK entered it (19); a request presenting another mandate is rejected with no transition. A successful validate consumes the nonce atomically with the transition it guards, after which $\text{fresh}(\eta(R)) = \perp$. The conjunct authorises re-asserts at the gate the authority established at pending_authority ; while the task's reservation stands, its budget conjunct reads $\text{tot}(M) \leq \text{cap}(M)$ and is discharged by the reservation itself.

The entitlement branch. An entitlement is held per client and carries the record

$$V = \langle \sigma_{\text{ent}}, \text{bal}, d_{\text{ent}}, Z_{\text{ent}} \rangle \in \text{Entitlement} \quad (11)$$

with covered scope $\sigma_{\text{ent}}(V)$, remaining balance $\text{bal}(V) \in \text{Money}$, expiry $d_{\text{ent}}(V)$ (with $\text{expired}(V) \Leftrightarrow \text{now} > d_{\text{ent}}(V)$) and $Z_{\text{ent}}(V)$ the un-consumed drawdown tokens minted against it. The accessor

$$\text{ent} : \text{Ent} \times C \rightarrow \text{Entitlement} \quad (12)$$

names the record method S draws against for a client. The map draw mints exactly one drawdown token $z = \text{draw}(q)$, with its own expiry, per entitlement-covered quote, against the one record the entitlement component selects to cover it (injective, server-minted, A9); one record can carry tokens for several of a client's tasks, and C10 allows one per task. z **earmarks no balance**. Unlike the mandate reservation of C9, which is subtracted from headroom at pending_authority , z is a single-use token that admits one draw for one task; the balance test is taken at admission, and what orders two tasks drawing on one record is C11; z orders nothing. The plan check is

$$\begin{aligned} \text{plancheck}(M, q, S) \Leftrightarrow & \text{scp}(q) \in \sigma_{\text{ent}}(V) \wedge m(q) \leq \text{bal}(V) \wedge \neg \text{expired}(V) \wedge \\ & \text{authorises}(M, q) \wedge \text{fresh}(\text{draw}(q)) \wedge \text{draw}(q) \in Z_{\text{ent}}(V) \wedge \\ & \neg \text{expired}(\text{draw}(q)), \quad V = \text{ent}(S, \text{issuer}(M)) \end{aligned} \quad (13)$$

The membership conjunct binds the draw to the record z was minted against: a request whose method names another of the client's records draws on a record that does not hold z , and is refused with no transition. An entitlement in one currency never covers a quote in another (§9.1). A successful plan check consumes z and decrements $\text{bal}(V)$ by $m(q)$ in the same atomic step, under the per-entitlement serialisation of C11; Corollary 1 is the invariant that step maintains.

The two predicates are the two admitting paths of one gate:

$$\text{admit}(q, S, M) \Leftrightarrow \text{validate}(\text{req}(q), S, M) \vee \text{plancheck}(M, q, S) \quad (14)$$

under the convention that a disjunct is $\perp$ where S lies outside its branch or where the quote's token is not the one that disjunct consumes, so a branch mismatch is a refusal and never an undefined guard. Each disjunct re-asserts authorises, is evaluated atomically with the transition it guards, and consumes a single-use token in that step.

Branch fixing and refusal. A quote fixes its branch: it carries either R or z . Within the payment branch the client chooses among the methods offered for R . **A refused activation request fires no transition: the task stays at the gate, its token stays fresh and its reservation stands**, so the client may present a credential for another offered method until an activation succeeds or the expiry of the token, or of the entitlement, takes the task to cancelled. Moving a purchase from the payment branch to an entitlement needs a replacement quote while the task is at quoted (REQUOTE), and after acceptance a new task. The model provides no gate-time conversion between branches.

9.5 Task state machine

Task lifecycle is a labelled transition system $T = \langle Q, \rightarrow, q_0, \text{Term} \rangle$ over the operator's canonical states,

$$Q = \{ \text{draft}, \text{quoted}, \text{accepted}, \text{pending_authority}, \underbrace{\text{pending_payment_or_plan_check}}_{\text{gate state } g^*}, \text{pending_expert_match}, \text{awaiting_expert_input}, \text{needs_client_clarification}, \text{in_review}, \text{ready}, \text{delivered} \} \cup \{ \text{cancelled}, \text{failed}, \text{compensation_recorded} \} \quad (15)$$

with $q_0 = \text{draft}$ and resting states $\text{Term} = \{ \text{delivered}, \text{cancelled}, \text{failed}, \text{compensation_recorded} \}$; $\text{Term}_{\text{exc}} = \text{Term} \setminus \{ \text{delivered} \}$. The three exception terminals have no out-edge, and delivered has exactly one, the dispute edge to compensation_recorded. The level map $\text{level} : (Q \setminus \text{Term}_{\text{exc}}) \rightarrow \mathbb{N}$ assigns

$$\begin{aligned} & \text{draft } 0 \cdot \text{quoted } 1 \cdot \text{accepted } 2 \cdot \text{pending_authority } 3 \cdot g^* 4 \\ & \text{pending_expert_match } 5 \cdot \text{awaiting_expert_input } 6 \cdot \text{needs_client_clarification } 6 \\ & \text{in_review } 7 \cdot \text{ready } 8 \cdot \text{delivered } 9 \end{aligned}$$

$$\Phi = \{ s \in Q \setminus \text{Term}_{\text{exc}} : \text{level}(s) \geq 5 \} \quad (16)$$

is the fulfilment region. The expensive side effects of fulfilment, which are not freely reversible, (committing expert sourcing, soliciting and recording paid micro-inputs, generating the artefact) occur only on transitions entering or internal to Φ . The safety argument rests on one structural property: **the only edge from $Q \setminus \Phi$ into Φ is the gate edge**

$$\text{pending_payment_or_plan_check} \xrightarrow{\text{admit}(q, S, M) = \top} \text{pending_expert_match} \quad (17)$$

which both admitting paths guard. pending_authority enters Φ on no edge: it resolves to g^* , to cancelled (refusal, decline or token expiry) or, under the fault family of Appendix A, to failed. needs_client_clarification is entered only from awaiting_expert_input.

The activation request. The gate transition is performed by an activation request presenting (q, S, M) (a request presenting no admission method is refused at step 4 and fires no transition, §6 steps 3 and 4), the request parameters β , an idempotency key $k \in K$ (keys namespaced per client, stable across retries under A7) and, on the payment branch, a credential. It is processed in this order.

1. *Digest.* The gate recomputes $H(\text{canon}(\beta))$ and rejects the request unless it equals $\text{ph}(q)$. A rejected request binds no key and writes nothing.
2. *Key.* A request not authenticated as the holder of the mandate its quote was issued under, or presenting another mandate, is rejected without effect, binds no key and observes nothing. Otherwise, where k is unbound, the task store binds it by an atomic compare-and-set $\text{CAS}(k : \perp \mapsto \text{taskId})$ (C8) and writes the activation record $\text{act}(k) = \langle k, \text{taskId}, o, t_{\text{act}} \rangle$ with $o = \text{pending}$. A request under a bound k proceeds only against the task bound to it.
3. *Authority.* At quoted the request fires *accept* and *begin authority check*; at *pending_authority* it resolves authority and escalation, and the edge into g^* reserves $m(q)$ against M (C9).
4. *Local conjuncts.* At g^* the operator evaluates authorises, freshness and expiry of the token, and the standing reservation.
5. *Provider.* For $S \in \text{Ext}$ the operator obtains the provider's outcome for the credential. A decline leaves authentic false. An indeterminate outcome leaves the task at g^* with the disposition stored (O3); a later definitive success completes step 6 if the token is still fresh and unexpired. A provider requirement for customer action leaves the task at g^* in the same way until the action completes (O3).
6. *Commit.* When *admit* holds, one atomic step consumes the token, commits the reservation, writes the task record with ρ_{pay} , sets $o = \text{admitted}$ and moves the task to *pending_expert_match*.

Requests presenting the same token are serialised from step 4 to step 6, so a second credential for an admitted quote is refused at step 4 before any provider call. A request refused at step 4 or 5 fires no transition (§9.4). o is set to refused in the same atomic step as any transition carrying the task to a terminal state before the gate edge fires. $\text{act}(k)$ records the disposition of the activation attempt and carries no financial reading. Under A6 and the injectivity of *canon*, a caller quoted for one set of parameters is not fulfilled for another, which is how C1 binds at the wire.

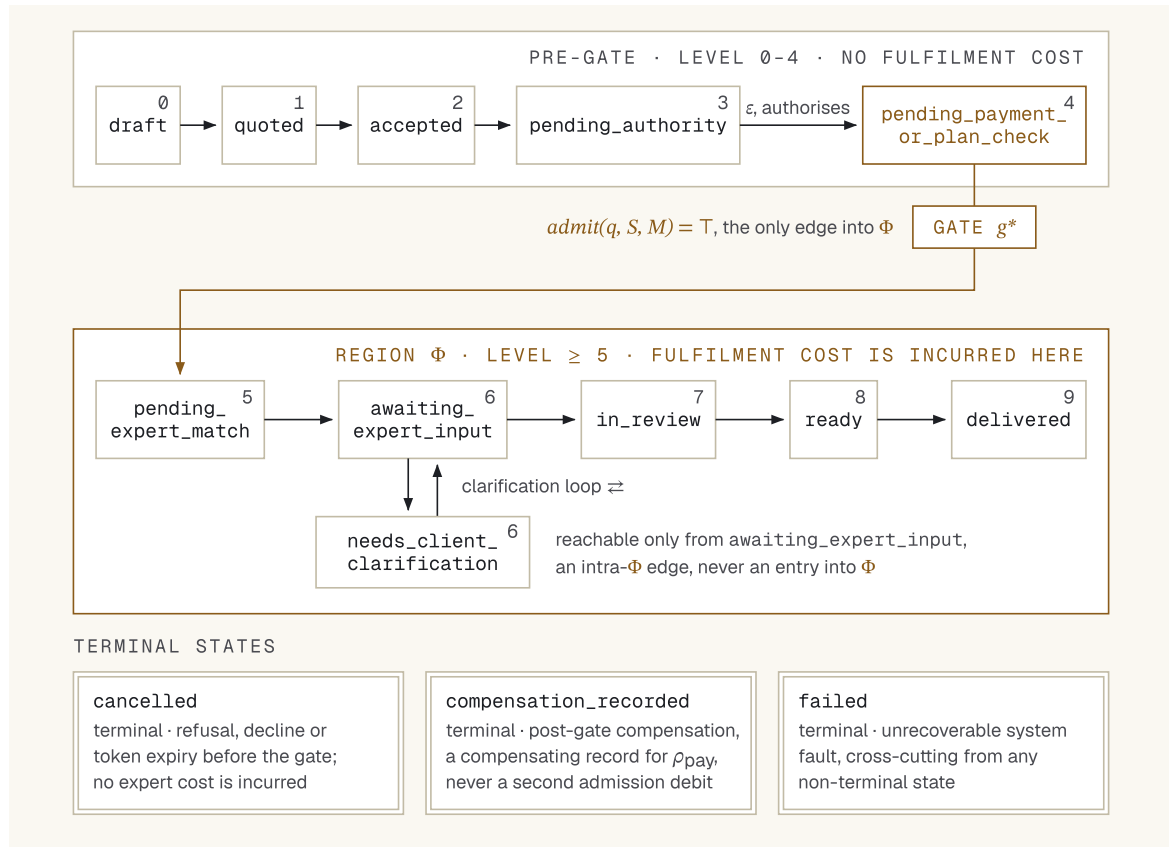


Figure 4. The task state machine. Pre-gate states sit at levels 0 to 4 and the fulfilment region Φ at level 5 and above; the only edge into Φ is the gate edge. The $\text{pending_authority} \rightarrow g^*$ edge labelled “ ε , authorises” stands for the two rows Appendix A prints, $\neg\varepsilon \wedge \text{authorises}$ and $\varepsilon \wedge \text{approved} \wedge \text{authorises}$, and the exception edges carry short labels; Appendix A gives every row.

9.6 Fulfilment, ledger, receipt

$$\text{ValidatedReq} = \{(q, S, M) : \text{admit}(q, S, M) = \top \text{ at the gate transition (17)}\}$$

$$F : \text{ValidatedReq} \rightarrow \text{Artefact}$$

Membership of *ValidatedReq* is fixed at the gate step. *F* is partial: an admitted task that finds no expert satisfying χ_M (A5), that times out or is withdrawn during clarification, whose artefact fails review, or that meets an unrecoverable fault ends at *compensation_recorded* or *failed*. Each such ending writes a compensating record

$$\text{comp} = \langle \rho_{\text{pay}}, \text{amt}_c, \text{reason} \rangle, \quad \text{amt}_c \leq m(q),$$

whose amount the deployment sets by policy for each reason; on an entitlement draw it adds amt_c back to $\text{bal}(V)$. *L* is the append-only ledger of entries $\ell = \langle \text{taskId}, e, \text{inputRef}, t_{\text{in}} \rangle$, and every entry bound to a task under *M* satisfies $\chi_M(\text{exp}(\ell)) = \top$ (C4). The delivery receipt and its signature are

$$\Omega = \langle M, \rho_{\text{pay}}, \text{taskId}, m(q_t), \text{artefactHash}, t_{\text{del}}, \text{mode} \rangle, \quad \sigma_\Omega = \text{Sign}(k_{\text{op}}, \text{canon}(\Omega)) \quad (18)$$

with *canon* extended to receipt tuples and injective on them, ρ_{pay} the admission reference (the credential accepted against *R*, or the consumed drawdown token *z*), $m(q_t)$ the frozen price of the task's in-force quote, in its currency, $\text{artefactHash} = H(\text{artefact})$, t_{del} the delivery time as the operator records it, $\text{mode} \in \text{Method}$ the admission method, and k_{op} the operator's signing key, whose public half the operator publishes. The delivery transition constructs *mode* from the task record

$$\text{task} : \text{TaskId} \rightarrow \text{Task}, \quad \text{task}(\text{taskId}) = \langle \text{taskId}, q, M, S, k, \rho_{\text{pay}}, m(q) \rangle,$$

written once at the gate transition and immutable thereafter. ρ_{pay} resolves by family: to a provider record for *Ext*, to the operator's billing record for *Bill*, and to the consumed reservation and its entitlement record for *Ent*. The receipt store $\text{rcpt} : \text{TaskId} \rightarrow \Omega \times \text{Sig}$ is written once, at the delivery transition.

verifyΩ takes a receipt, its signature and a delivered artefact. It first verifies σ_Ω under the operator's published key, then runs five checks: (1) *H* over the artefact equals *artefactHash*; (2) ρ_{pay} resolves against the admission record its family names, and that record's amount and currency equal the receipt's $m(q_t)$; (3) *M* resolves against the mandate registry; (4) $\text{task}(\text{taskId})$ is defined and carries the receipt's *M*, ρ_{pay} and $m(q_t)$; (5) *mode* equals the *S* of that record. A receipt whose signature verifies establishes what the operator attested at delivery, wherever it is presented. Check (1) needs nothing else. Checks (2) to (5) need the mandate registry entry, the admission record and the task record, and the retrieval contracts of §7 return none of those records: they return the artefact, the receipt and a task status that reports the task's state and dispositions, without the fields checks (2) to (5) compare. A verifier outside the operator therefore needs an authenticated audit interface to those records, which a deployment provides and this paper does not specify; without it, an external party can verify the signature and the artefact hash, and checks (2) to (5) rest on records the operator supplies (A8). Neither the signature nor the five checks is evidence that funds moved: check (2) binds the receipt to the amount the operator recorded as admitted, with no payment figure in it.

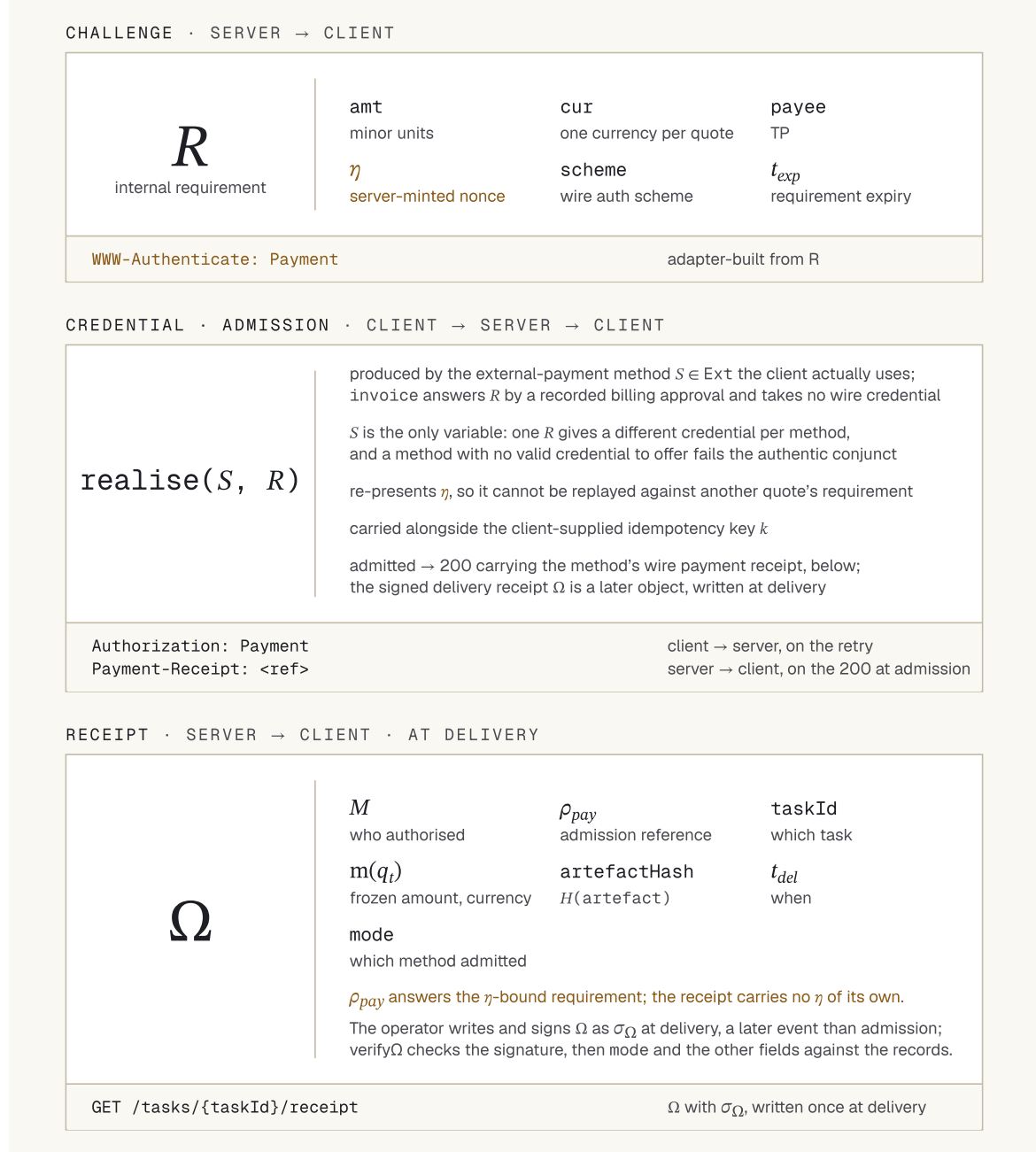


Figure 5. The three objects of the exchange field by field: the internal payment requirement R ; the credential that answers one of its challenges, produced by an external-payment method, with the method's wire Payment-Receipt returned on the 200 at admission; and the signed delivery receipt Ω of (18), written at delivery with its seven fields, which reaches the admission through ρ_{pay} and carries no nonce of its own.

9.7 Well-formedness constraints

C1 (Price-freeze). a task activates only at $m(q)$ of its issued quote; any change to (p, ρ) requires a new quote and a new admission token, and an activation request is admitted only where $H(\text{canon}(\beta)) = \text{ph}(q)$.

C2 (Scope). $\text{scp}(q) \in \sigma_{\text{auth}}(M)$.

C3 (Deadline). $\text{dl}(q) \leq d_{\text{lim}}(M)$.

C4 (Expert-criteria). $\forall \ell \in L$ for the task, $\chi_M(\text{exp}(\ell)) = \top$.

C5 (Authority and budget). $\text{cur}(m(q)) = \text{cur}(\text{cap}(M))$ and $\text{tot}_q(M) + m(q) \leq \text{cap}(M)$.

C6 (Escalation). where $\varepsilon_M(q) = \top$, the task reaches the gate only with $\text{approved}(M, q)$ recorded, and it is never advanced automatically.

C7 (Single requirement). at most one un-consumed requirement R exists per task at any time.

C8 (Idempotency-key uniqueness). $\text{CAS}(k : \perp \mapsto \text{taskId})$ succeeds at most once per k and at most once per task, and runs only for a request whose parameter digest matches its quote. The key binds to the pair (quote identifier, parameter digest). Every later request under k addresses the originally bound task; a request under k whose digest differs from the bound request's, *or which names another quote*, or which presents another mandate, is rejected without effect, and a request under another key for a task that already carries one is rejected without effect. The quote identifier is in the binding because $\text{ph}(q)$ covers the parameters while leaving `quote_id` out, so two quotes for one product with identical parameters share a digest and a key reused across them would otherwise be served as a replay of the first.

C9 (Serialised mandate spend). the spend record of a mandate is maintained under a per-mandate serialisation. On the `pending_authority` $\rightarrow g^*$ edge the operator evaluates **C5** against committed plus reserved spend and, in the same atomic step, reserves $m(q)$, so two activations under one mandate cannot both read the pre-reservation figure. The reservation is committed to $\text{spent}(M)$ at admission and released on every other exit from the gate. An unlock (§9.9) tests **C5** and commits $m(q)$ in one atomic step. A committed admission debit is never returned to $\text{res}(M)$ or subtracted from $\text{spent}(M)$, so $\text{cap}(M)$ bounds the gross admitted amount over the mandate's life and a compensated admission still counts against the cap; restoring headroom on compensation is a policy outside the model, distinct from restoring $\text{bal}(V)$.

C10 (Single drawdown token). at most one un-consumed drawdown token z exists per task at any time, consumed atomically with the gate transition it admits.

C11 (Serialised entitlement draw). the balance record of an entitlement is maintained under a per-entitlement serialisation. A plan check reads $\text{bal}(V)$ and decrements it by $m(q)$ in one atomic step under that serialisation, so two draws on one record cannot both read the pre-decrement figure. The restoration of amt_c on a compensating record is taken under the same serialisation.

C2, **C3** and **C5** are conjuncts of authorises; **C1** is enforced by req and draw at issue and by the digest check; **C7** and **C10** by the minting maps and the task engine; **C4** by the ledger invariant; **C6** by the guard at pending_authority; **C8** by a unique index on keys; **C9** by a per-mandate reserve-and-commit; **C11** by a per-entitlement serialisation of the balance record. Together they make the results below statements about every well-formed run. q_t is the unique in-force quote of task t .

9.8 Properties

An *admission debit* is the admission record keyed to the single token consumption the gate transition performs: η on the payment branch, z on the entitlement branch. At most one is created per admitted task. A *charge*, in this paper, is a provider event: the authorisation, capture or settlement of funds by the selected method's provider, which lies outside the model and is governed by O2 to O4 and by that method's own idempotency. T2, T3 and T5 concern admission debits and establish nothing about the uniqueness of provider charges.

Within the model, and subject to **A1** to **A9** (§13) and **C1** to **C11**, T1 to T4 below hold by construction and T5 by induction. T5 quantifies across the tasks of a mandate and is stated first; T1 to T4 are structural facts over $\rightarrow$ of T . An execution trace is a finite path $q_0 \rightarrow^* s$ respecting the guards of §§9.4 and 9.5. The validation supplement checks T1, T3, T5, **Corollary 1** and the drawdown token's record binding exhaustively on bounded instances (§12).

T5 is stated over a configuration

$$\Gamma = \langle \text{st}, \text{qt}, \text{md}, \text{sp}, \text{bal}, N, Z, K, \text{Act}, \text{Apr} \rangle$$

of task states, in-force quotes, each task's mandate, per-mandate committed and reserved spend, entitlement balances, un-consumed nonces and drawdown tokens, bound keys, activation records and recorded escalation decisions. The initial configuration Γ_0 has every component empty. Configurations are built by the introduction rules

$$\begin{array}{lll}
 \text{MAND} : & M \notin \text{dom}(\text{sp}) & \longrightarrow \text{sp} := \text{sp} \cup \{M \mapsto (0, 0)\} \\
 \text{ENT} : & V \notin \text{dom}(\text{bal}) & \longrightarrow \text{bal} := \text{bal} \cup \{V \mapsto b_0(V)\} \\
 \text{TASK} : & t \notin \text{dom}(\text{st}), M \in \text{dom}(\text{sp}) & \longrightarrow \text{st} := \text{st} \cup \{t \mapsto q_0\}, \text{md} := \text{md} \cup \{t \mapsto M\} \\
 \text{QUOTE} : & \text{st}(t) = q_0 & \longrightarrow \text{st}(t) := \text{quoted}, \text{qt}(t) := q, \text{tok}(q) \text{ minted into } N \text{ or } Z \\
 \text{REQUOTE} : & \text{level}(\text{st}(t)) = 1 & \longrightarrow \text{qt}(t) := q', \text{tok}(q) \text{ withdrawn}, \text{tok}(q') \text{ minted}
 \end{array} \tag{19}$$

MAND enters a mandate at zero spend and ENT an entitlement at an opening balance $b_0(V)$ supplied by the deployment. TASK is the only rule that writes md or extends dom(st), so a task never changes mandate. QUOTE fires with the *submit* edge. Every drawdown token, from QUOTE or REQUOTE, is minted against the one record the entitlement component selects to cover its quote, so Z is the union of the sets $Z_{\text{ent}}(V)$. REQUOTE is enabled only at quoted, because escalation is resolved once at pending_authority on the in-force quote and acceptance is given to a particular quote; it withdraws the superseded token without consuming it. The global step relation $\Rightarrow$ is generated by these rules, the task transitions of §9.5 with their global effects (token consumption, the bal(V) decrement and its restoration, writes to Act), the key-binding compare-and-set, the recording of escalation decisions, the unlock step of §9.9 and the spend operations of **C9**. Projecting a $\Rightarrow$ -trace onto one task's transitions returns a trace of T . A lifecycle read (status, result, receipt retrieval) takes no step of $\Rightarrow$.

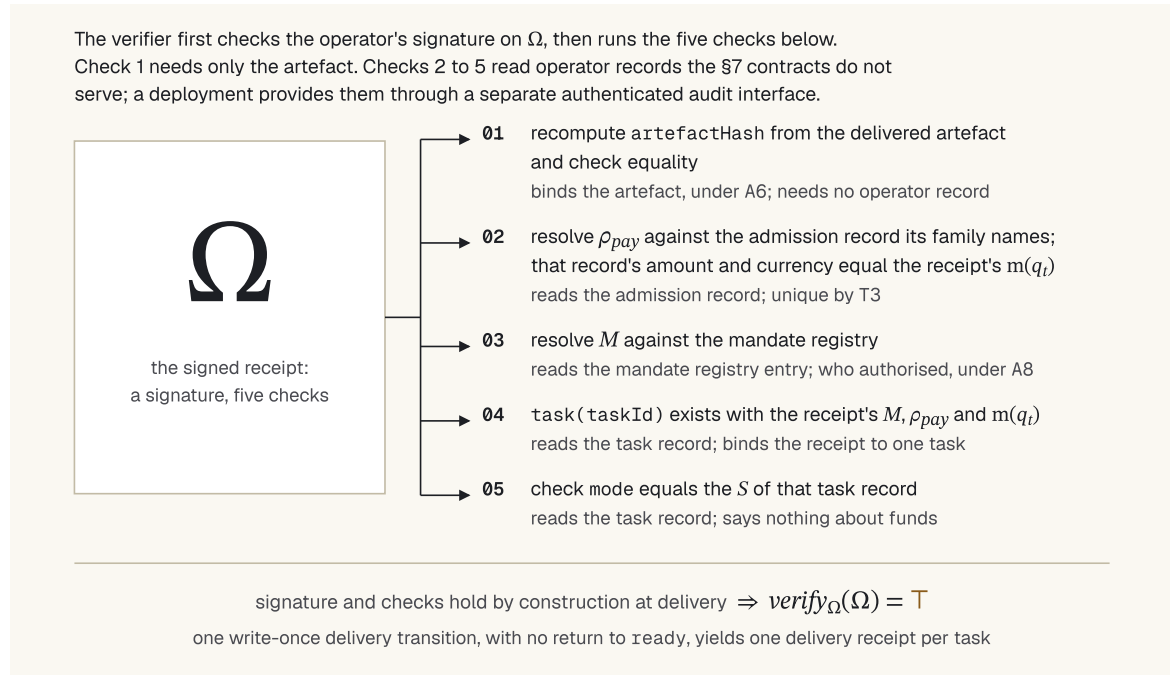


Figure 6. Receipt verification: the operator's signature on Ω , then five checks, each marked with the record it reads (§9.6). Checks 2 and 4 compare the resolved record's amount and currency with the receipt's $m(q_t)$.

Trust boundary. The properties are guarantees against a misbehaving or compromised client side (an agent that replays a consumed token, retries without a stable key, presents a forged or expired credential, or attempts spend outside its mandate) over a network with at-least-once delivery. They are not guarantees against the operator: True Primary runs the task engine, the token stores, the entitlement records, the spend record, the ledger and the receipt store, so gate ordering and spend serialisation are invariants it enforces on its own execution (A3), which no external party can compel. What an external party can check is the signed receipt, and the records behind checks (2) to (5) to the extent the operator grants access to them. **The buyer also holds a per-token bound.** On the payment branch a shared payment token carries a currency, a maximum amount and an expiry, which the agent that issues it sets, matching the maximum amount to the transaction's total [22]; the provider enforces those limits on each token, outside the operator. They bound each token separately and say nothing about the total admitted across several tokens under one mandate. That aggregate bound is T5, and it rests on the operator's spend record and its serialisation.

Theorem 1 (Budget-cap soundness). *In every reachable configuration Γ and for every mandate M , $\text{spent}(M) \leq \text{cap}(M)$, including where one mandate authorises several activations concurrently.*

Proof. We show $\text{tot}(M) \leq \text{cap}(M)$ is invariant, by induction over $\Rightarrow$. By the currency conjunct of C5, every quote reserved or committed under M carries the currency of $\text{cap}(M)$, so both sums are defined. (a) In Γ_0 nothing is reserved or committed, so $\text{tot}(M) = 0 \leq \text{cap}(M)$. (b) By C9 the only steps that change $\text{spent}(M)$ or $\text{res}(M)$ are reservation, commit, release and the unlock's test-and-commit, all taken under the per-mandate serialisation, so the steps touching one M are totally ordered. A reservation of $m(q)$ occurs only where C5 holds against committed plus reserved spend; q then holds no standing reservation, so the test reads $\text{tot}(M) + m(q) \leq \text{cap}(M)$, and test and reservation are one atomic step. The unlock's test-and-commit is the same test followed in the same step by an increment of $\text{spent}(M)$. A commit moves $m(q)$ from $\text{res}(M)$ to $\text{spent}(M)$ and leaves $\text{tot}(M)$ unchanged; a release decreases it. Every step preserves the invariant; steps outside the spend operations write no component of sp . (c) $\text{res}(M) \geq 0$, so $\text{spent}(M) \leq \text{tot}(M) \leq \text{cap}(M)$.

C5 alone does not yield this: read as a test on $\text{spent}(M)$ it is a read-then-check on shared state, and two activations under one mandate could each pass against the same figure. C9 closes that and is the only constraint quantifying across the tasks of a mandate. The commit side is exact: by Structural fact 3 at most one admission debit of $m(q_r)$ arises per task, so each admission debit commits exactly one reservation. Structural fact 3 rests on token consumption alone and Structural fact 2 on C8 and Structural fact 3; neither uses this theorem. The argument is indifferent to the admitting branch. The mandate reservation bounds spend across the tasks of one mandate; the drawdown token bounds one task to one draw, and C11 with Corollary 1 bounds the balance across the tasks drawing on one entitlement record. An entitlement-admitted task takes all three. $\square$

Corollary 1 (Entitlement-balance soundness). *In every reachable configuration Γ and for every entitlement V , $\text{bal}(V) \geq 0$, including where several tasks draw on one record concurrently.*

Proof. ENT enters V at an opening balance $b_0(V) \geq 0$. By C11 the only steps of the model that change $\text{bal}(V)$ are the plan-check decrement and the restoration of amt_c on a compensating record, both taken under the per-entitlement serialisation, so the steps touching one V are totally ordered. A decrement of $m(q)$ occurs only where the conjunct $m(q) \leq \text{bal}(V)$ of (13) holds, and test and decrement are one atomic step, so the balance after the step is $\text{bal}(V) - m(q) \geq 0$. A restoration adds $\text{amt}_c \geq 0$. Every step preserves the invariant, and steps outside these two write no entitlement balance.

C11 is what yields this, and the balance conjunct of (13) alone does not. Read as a test on $\text{bal}(V)$ it is a read-then-check on shared state: two plan checks drawing on one record could each pass against the same pre-decrement figure and then each decrement, taking the balance negative by up to the smaller of the two prices. z does not close it, because draw mints z per quote and C10 bounds one task to one draw; z earmarks no balance and orders nothing between two tasks. C11 is to the entitlement record what C9 is to the spend record, and they are the only two constraints quantifying across tasks. The bound is on the record, with the mandate outside it: two mandates of one client drawing on one V are ordered by C11 and by nothing in C9. $\square$

Structural fact 1 (Gate-before-fulfilment safety). *In every reachable trace, each side effect of F follows a successful admission, $\text{admit}(q, S, M) = \top$, in the same atomic transition or in an earlier transition of the same trace.*

Proof (by construction). (a) $q_0 = \text{draft}$ has level $0 < 5$, so $\text{draft} \notin \Phi$. (b) The exception terminals have no out-edges, and the fault edges all target $\text{failed} \notin \Phi$. Enumerating Appendix A, the set $\{(s \rightarrow s') : s \notin \Phi, s' \in \Phi\}$ is the singleton $\{g^* \rightarrow \text{pending_expert_match}\}$ (17), guarded by $\text{admit} = \top$. pending_authority resolves only to g^* , cancelled or failed; the gate's other exits go to cancelled or failed; a refused request fires no edge. (c) Every other edge into Φ has its source in Φ . (d) Hence any trace reaching Φ contains the gate step under $\text{admit} = \top$. Side effects of F occur only on transitions entering or internal to Φ , so each follows that step or occurs within it after its guard held. The step consumes the task's token, so no trace can leave Φ and re-enter without a fresh token, which only a new quote mints (C1, C7, C10). $\square$

Structural fact 2 (Idempotency under a request key). *For any multiset of activation requests sharing an idempotency key k : at most one activation and at most one admission debit occur; among requests under k authenticated as the holder of the bound task's mandate and presenting that mandate, a request is a replay exactly when it names the quote and carries the parameter digest of the request that bound k , and every replay addresses the task bound to k ; once the activation resolves, every replay observes the same activation record $\text{act}(k)$ and, after delivery, the same receipt $\text{rcpt}(\text{taskId})$; any other request under k is rejected without effect.*

Proof. A request whose digest differs from its quote's is rejected before the key step and binds nothing (§9.5, step 1), and one not authenticated as the holder of its quote's mandate, or presenting another mandate, is rejected at step 2 before any binding. The first remaining request under k runs $\text{CAS}(k : \perp \mapsto \text{taskId})$, which succeeds and writes $\text{act}(k)$ with $o = \text{pending}$; every later request under k finds the binding. One not authenticated as the holder of that task's mandate, or presenting another mandate, is rejected without effect at step 2 and observes nothing; among the rest, by C8, one that names the bound quote and carries the bound digest addresses that task, while one naming another quote or carrying another digest is rejected without effect. While $o = \text{pending}$ a replay is evaluated at the task's current state, and a refused evaluation fires no transition. An activation is the gate edge, which consumes the task's single token, so by Structural fact 3 at most one fires, and at most one admission debit arises. Once the edge fires ($o = \text{admitted}$) or the task reaches a terminal state first ($o = \text{refused}$), later replays observe that record, and after delivery the same Ω , without re-invoking F . On the entitlement branch read z for η ; no other step changes. $\square$

Structural fact 3 (Single admission-token consumption). *For any task t , at most one admission token is consumed, so at most one admission debit of $m(q_t)$ arises. A later compensation writes a compensating record against ρ_{pay} and raises no second admission debit. The provider's own capture lifecycle is outside the model (O2 to O4).*

Proof. Payment branch. (a) req issues exactly one R per quote with a unique nonce, and by C7 at most one un-consumed R is in force per task. (b) Consumption happens only in the commit step of §9.5, atomically with the gate transition (A9); a refused request consumes nothing, and after consumption $\text{fresh}(\eta) = \perp$, so no second commit against the same R can pass. (c) A second R for t needs a new quote, and supersession is enabled only at quoted, before the gate, so a consumed R is never superseded. (d) Concurrent requests presenting the same R are serialised through the commit step; at most one consumes η , and every other observes $\text{fresh}(\eta) = \perp$. Hence at most one R is ever consumed for t . *Entitlement branch.* The same four steps hold with z for η , draw for req and C10 for C7; the commit step also decrements $\text{bal}(V)$ once, under C11 (Corollary 1), and by (13) the record it decrements is the one the token was minted against. A later transition to `compensation_recorded` or `failed` writes `comp` against ρ_{pay} , restores amt_c to $\text{bal}(V)$ on an entitlement draw, and consumes no token. $\square$

Structural fact 4 (Operator-attested receipt integrity). *Every task in `delivered` has a unique signed delivery receipt $\text{rcpt}(\text{taskId})$, and $\text{verify}\Omega$ holds for it and its delivered artefact, binding the mandate, the admission, the task, the frozen price, the artefact and the admission method; checks (2) to (5) run against records the operator makes available.*

Proof. The only in-edge to `delivered` is `ready` $\rightarrow$ `delivered`, guarded by Ω and σ_Ω constructed and written, and no edge returns a delivered task to `ready`, so each delivered task carries exactly one receipt. The signature holds by construction under A6. (1) `artefactHash` is computed over the artefact delivered. (2) ρ_{pay} is written at the gate transition with the admission record its family names, and that record carries $m(q_t)$, which C1 froze at quote issue and no later step changes. (3) $M = \text{md}(\text{taskId})$ was entered by MAND. (4) Ω copies M , ρ_{pay} and $m(q_t)$ from the immutable task record. (5) `mode` is constructed from the S of that record. Each lookup returns what was written, under A8. The result concerns delivered tasks; an admitted task that ends without delivery carries a compensating record and no receipt. $\square$

Corollary 2 (Operator-side integrity). *T1 to T5 hold together: no expert effort is incurred before authorised admission (T1); a request key yields at most one activation and one admission debit (T2); a task consumes at most one admission token (T3); every delivered artefact carries a unique signed receipt that checks against the operator’s own records, which is attestation by the record keeper, with no independent party attesting it (T4); and the admission debits under one mandate never exceed its cap in total, however many run at once (T5). Corollary 1 adds the entitlement side of the same discipline: a prepaid balance never goes negative, however many tasks draw on it at once. Together they are the discipline an expert-access business already applies by hand, written so that a machine can apply it.*

9.9 Paid unlock of a stored artefact

A stored artefact, such as a transcript summary, is sold by unlock. An unlock quote q has the unlock product, the artefact identifier as its parameters, a frozen price and one admission token on either branch. For each pair of artefact and holder the model keeps two states, `locked` and `unlocked`, and one edge

$$\text{locked} \xrightarrow{\text{admit}(q, S, M) = \top} \text{unlocked}$$

taken by `POST /paid-artifacts/{artefactId}/unlock`. The model fixes the edge's budget step and nothing more of its order: the edge tests [C5](#) and commits $m(q)$ in one atomic step under the per-mandate serialisation ([C9](#)), the same step that consumes the token, with no reservation and no fulfilment region. Where that step sits relative to the provider call is not fixed, so the unlock is not specified as a complete transaction and lies outside the implementation contract of §10. The edge writes, in its atomic step, the grant

$$G(\text{artefactId}, \text{holder}(M)) = \langle \text{artefactId}, M, S, k, \rho_{\text{pay}}, \text{artefactHash}, t_{\text{grant}}, \text{mode} \rangle \quad (20)$$

signed with k_{op} as Ω is. `GET /paid-artifacts/{artefactId}` returns the artefact only to a caller authenticated as a holder with a grant for it, or as the holder of record of the task that produced it. **Escalation does not carry over implicitly.** Step 3 of §9.5 is the only place ε_M is resolved as a transition, and the unlock edge has no such step, so an unlock of a quote for which $\varepsilon_M(q) = \top$ reaches `unlocked` with no recorded human decision unless the deployment adds one. A deployment that applies escalation to unlocks adds an approval guard to admit on this edge and records the decision as [C6](#) requires; this paper states the extension without it and does not prove the escalation discipline over it. The grant also carries no quote or price reference of its own: a deployment binds the immutable quote identifier and the frozen price into (20) before relying on the price-freeze discipline for unlocks. Subject to those two, and to a complete transaction that keeps that atomic step, the four structural facts and T5 carry over with the grant in place of the task record and receipt. No artefact is returned before its grant is written, since the grant is the only authority for retrieval (T1). A key binds to one unlock, and the edge fires at most once because it consumes the token (T2, T3). The grant binds the artefact hash, the mandate, the admission and the method under the same signature and lookups as Ω (T4). T5 holds because the unlock's spend step is one atomic test-and-commit under the per-mandate serialisation. A deployment that offers the unlock specifies that transaction in full, with the approval guard and the price binding, and re-establishes these results for it.

10 Safety & implementation constraints

The architecture's safety rests on one discipline, gate before fulfilment, implemented defensively because the wire scheme its adapters use is an unadopted Internet-Draft. T1 to T5 of §9.8 are its acceptance criteria, T5 under the no-restoration baseline of O8. The constraints below bring a deployment within the model, and obligations O1 to O8 cover what the model does not carry.

Gate order. The six-step order of the activation request (§9.5) is the implementation contract for paid tasks: digest before key, local conjuncts before any provider call, and one local atomic commit. The paid unlock of §9.9 is an extension outside it. **The request that draws the 402 writes pre-gate task state.** On the payment branch the activation request arrives first without a credential and, unless it selects invoice (§6), is refused at the gate with the 402 that carries the challenges; it writes the key binding, the task at the gate, the adapter’s mapping of each challenge to η (O7) and the reservation the challenge’s amount is honoured against. It makes no provider call and incurs no fulfilment cost (T1), and every exit from the gate other than admission releases the reservation (O5) and consumes no token. The pinned draft states that “The unpaid request that triggers a 402 challenge MUST NOT modify server state beyond recording the challenge itself” [3, §11.4]. The activation of §6 writes more than that by design, so that headroom is held before any challenge issues (§6), and the paper claims no conformance to that section. A deployment that needs conformance establishes the necessary state through an authenticated preflight instead, and bounds the resource cost of unpaid reservation. admit is evaluated in full and fails closed on any conjunct; an authentic credential under a mandate that does not authorise the quote is refused with no transition, as a forged one is. The two differ on the wire: a mandate refusal answers 403 and a failed credential 402 with fresh challenges (§6).

Idempotency keys and tokens. The key is the mechanism behind T2, under the retry contract of §7, and the single-use token the mechanism behind T3. A credential for a consumed token cannot drive a second admission. Both are enforced at the gate, upstream of Φ , so a client can retry the task endpoints over an unreliable network.

Receipt signing. Every delivery writes Ω and σ_Ω once to the receipt store. A deployment publishes the verification key at a stated URL, indexed by the `key_id` each receipt carries (Appendix B.4), and keeps every key it has signed with verifiable for as long as its receipts are relied on.

Defensive implementation against a draft. Pin the implemented revision of the “Payment” scheme (§12) and version header names and challenge shapes against it. Make no single payment method load-bearing: invoice and the three entitlements stay admission methods beside card, wallet, SPT and stablecoin. The draft defines how a challenge and credential travel; the operator’s own task engine defines when fulfilment may begin, and a revision of the draft changes the adapters, which O7 binds to R , and leaves the gate unchanged.

Challenge expiry is quote expiry. Every challenge for a quote carries R ’s `expires_at` as its expires (Appendix B.1), so both lapse at one instant. Before it, a refused credential draws a fresh 402, as the draft’s status table gives [3, §4.2]. After it, the draft answers an expired challenge with a fresh one and a `payment-expired` problem [3, §4.2]; this model has retired the token with the quote, a fresh challenge would need a new quote (C1), and it answers 410 with the task state (§6), a second departure from the draft beside the one in the gate order above. At the instant itself the model counts the token live, since `expired( $\omega$ )` holds only once now $> t_{\text{exp}}(\omega)$ (§9.2), while the draft bars clients from presenting a credential for an expired challenge [3, §5.1.2] and fixes no convention for the instant; a deployment states the one it uses.

Deployment obligations. A deployment sits inside a wider system than the model carries. These obligations are what that system owes at the boundary. Each names the record or test that discharges it, and no proof in §9 uses them.

- **O1 · Eligibility and credit, before a quote.** Nothing in §9 prevents a quote offering a method the client, entity, jurisdiction or risk profile cannot use, and A4 assumes that determination. A deployment offers a method for R only where its provider presents in $\text{cur}(R)$; sets a credit limit per client before offering invoice and, because the model keeps no ledger of a client's receivables across mandates, counts outstanding receivables and standing reservations across them and reserves credit atomically before admitting an invoice wherever it relies on that limit as a bound; and decides, as payer-identity policy, whether the funding instrument presented against R must belong to issuer(M). *Discharged by* a recorded determination per method and per client, re-checked on a stated cadence.
- **O2 · Provider idempotency.** *For each external-payment method:* T2 and T3 hold at the operator's surface and assert nothing about the provider's execution of the call that follows; retrying a timed-out provider call can create two provider-side effects. The adapter keeps one purchase identity per admission token and one operation identity per provider attempt, and reuses an operation's provider idempotency key only for a retry with identical parameters. A replacement credential after a refusal starts a new attempt, with a new operation identity, only once the prior attempt has failed definitively or been reconciled; an attempt whose outcome is unknown is resolved first (O3). *Discharged by* a stored mapping from each token to its purchase identity and from each attempt to its provider reference, and serialisation of requests presenting the same token.
- **O3 · Indeterminate outcomes.** A provider call that returns neither success nor failure leaves the task at the gate. While the outcome is indeterminate the deployment makes no representation, to a client or in its records, that funds have moved. A later success completes admission only if the token is still fresh and unexpired; otherwise the provider effect is voided or refunded under O4. A provider that reports that the customer must act before the credential can complete is handled in the same way: the task stays at the gate with an action-required disposition and the provider's action reference, answered 202; admission resumes only after completion is confirmed, by re-running steps 4 to 6 of §9.5, and an expiry or withdrawal in the meantime ends the task at cancelled and sends any provider effect to O4. On a card payment through a shared payment token Stripe may trigger 3D Secure and redirect the customer to the bank [22], so a purchase the mandate authorises can still need its principal present. *Discharged by* a stored disposition, a bounded resolution procedure, and a status contract that reports it.
- **O4 · Reconciliation.** Only the gate reads a provider's outcome, through authentic; no guard of Appendix A tests one after admission, so the operator's records and the provider's can disagree without any transition detecting it. *Discharged by* a run on a stated cadence that pairs each admission with its provider reference, each compensating record with the provider's refund outcome, and each unpaired provider effect (a second transfer for an admitted quote, or a success after the token expired) with a refund, through an exception queue.
- **O5 · Expiry timers.** The time-driven rows of Appendix A fire only if a deployment runs their timers, and a reservation standing at the gate is released when the task leaves the gate by its expiry, by the client's withdrawal (§7) or by a fault exit. *Discharged by* an expiry job on a stated cadence, a count of tasks past their token's expiry, and a withdrawal endpoint a client can call at the gate.
- **O6 · Role and perimeter.** Whether an operator falls within a regulatory perimeter is a determination for that operator and its advisers on its own funds flows, and every method-specific duty that follows rests with the deployer. *Discharged by* that determination, recorded before a method is enabled in production.

- **O7 · Adapter binding.** Every wire challenge an adapter issues for one quote maps to that quote's R and η ; a credential verified against any of them is presented to the gate as the credential against R ; and a later credential for another challenge of the same quote is refused at the local-conjunct step because η is consumed. *Discharged by* a stored mapping from each challenge identifier to η and a conformance test in which credentials for two challenges of one quote yield one admission.
- **O8 · Headroom restoration.** Under C9, $\text{cap}(M)$ bounds *gross* admitted amounts over a mandate's life, so two compensated purchases of half the cap exhaust it with nothing delivered. A deployment may, by policy, credit a compensating record's amt_c back to the mandate's headroom by a recorded step taken under the per-mandate serialisation. **That step changes the quantity the cap bounds, and T5 does not carry over to it.** Once a compensation lowers $\text{spent}(M)$, a cap of c admits more than c in gross over a sequence of compensated and restored admissions: admit c , compensate c , restore c , admit c again, and the ledger reads c while $2c$ has been admitted. A deployment that restores headroom is therefore running a different accounting model, and it states which quantity its cap bounds, separates gross admissions from compensation credits, and carries its own invariant and its own proof over repeated compensation cycles. A deployment that does not adopt the policy tells principals that a mandate is a gross-attempt budget, because that is not what the phrase budget cap leads them to expect. *Discharged by* a stated policy of not restoring, or, where restoration is adopted, a separate gross-admission ledger with a replacement invariant proved over it.

11 Failure modes

The table sets out the failure modes of the admission and fulfilment path by position relative to the gate. Each row names where the failure is detected and how it is handled, and is classed as a model transition (M), a deployment obligation (O) or a policy control (P). The table asserts no eventual resolution.

Failure	Detected at	Handling	Class
<i>Before the gate</i>			
Parameter digest differs from the quote	activation request	rejected before the key binds; 422; no state change	M
Key reused with other parameters, another quote or another mandate	activation request	rejected without effect; 422	M
Mandate violation (holder, scope, currency, budget, deadline)	quote	refused; 403; no token, no challenge; the task stays at draft	M
Mandate stopped authorising after the quote	pending_authority, gate	refused; 403; to cancelled from pending_authority; at the gate no transition, and a pre-funded transfer already made is paired for a refund under O4	M
Pre-funded transfer for a quote refused at the gate after its challenge was drawn	reconciliation	paired as an unpaired provider effect for a refund under O4; the model states no interval	P
Escalation required	pending_authority	waits for a recorded decision; 202; token expiry exits	M
Token expired	quoted, pending_authority	to cancelled	M

Failure	Detected at	Handling	Class
Method unavailable	quote	not offered, or not presented; the task waits within its token's life	P
Mandate revoked before expiry	any	outside the enumerated path; A1 assumes mandate status holds	P
<i>At the gate</i>			
No credential, invoice not selected, and the quote offers a method in <i>Ext</i>	gate	402 with fresh challenges; no transition; reservation standing	M
No credential, invoice selected (§6 step 3), and no approval is recorded	gate	202 with a status link; no transition	M
Credential malformed, refused or declined	local conjuncts, provider	402 with fresh challenges; no transition	M
Replay of a consumed token, or a second credential for an admitted quote	local conjuncts	refused as an admission; under the bound key it is a replay, answered with the recorded outcome (§6); nothing consumed (T3)	M
Provider outcome indeterminate	provider	no transition; disposition stored; 202	O3
Entitlement short, out of scope, or drawn on a record without the quote's token	plan check	refused; 403; no challenge; no transition, token fresh, reservation standing	M
Token or entitlement expired at the gate	gate	to cancelled; reservation released	M, O5
Client withdraws at the gate	gate	to cancelled; reservation released; no token consumed	M, O5
<i>After the gate</i>			
No qualifying expert	pending_expert_match	to compensation_recorded; the compensated admission still counts against cap(<i>M</i>) (C9)	M
Clarification timeout or withdrawal	needs_client_clarification	to compensation_recorded	M
Release review withholds the artefact	in_review	unrecoverable-defect edge to compensation_recorded	P
Provider reversal or failed capture after admission	provider	no guard tests it; the admission stands; the unpaid quoted principal is at most $m(q)$ for the task, and fulfilment cost, fees and what is recovered lie outside that bound; recovery runs under the provider's terms	O4
Second transfer on a pre-funded method for an admitted quote	provider	paired under O4 for a refund to the paying wallet [23]; for one extra transfer the client's excess outlay is $m(q)$ until the refund completes, which the model does not establish	O4
Dispute upheld	delivered	to compensation_recorded	M
System fault	any	to failed; a compensating record after the gate	M
Delivery without a receipt	delivered	unreachable (T4)	M

Outside that path lie provider-side duplicate capture and chargeback (O2 to O4), availability attacks, and operator or expert adversaries, which the trust boundary of §9.8 excludes. Confidentiality, conflicts and material non-public information are compliance controls run at the control points of §14.

Capture sequencing. Admission precedes fulfilment, so an admitted task can find no expert. **Under the card intent this paper pins, a card credential is charged when the server accepts it:** the intent is defined as immediate processing, with the server returning success after authorisation approval while final settlement is pending [12], and the `intent` parameter must name a registered entry in an IANA registry the “Payment” draft states is initially empty [3]. The pinned draft’s own §7.3 prints `intent="authorize"` in an example alongside `intent="charge"`, and each method document defines the intents it supports; the card and Stripe methods pinned here specify immediate processing, and no implemented authorise-only method for a shared payment token is established here. Over the MPP exchange as specified, the card path therefore *charges at admission*, days before delivery, and a non-delivery is a refund under O4 instead of a capture that never happens.

Authorising at the gate and capturing at delivery is available only where a deployment creates the card `PaymentIntent` itself, with manual capture, outside the MPP intent. Stripe’s shared-payment-token documentation states nothing about manual capture for such tokens, so this paper makes no claim about whether that combination is supported. Two provider windows bound it if it is: Stripe gives a Visa authorisation 4 days 18 hours when merchant-initiated and 7 days when customer-initiated, and classifies a transaction as one or the other “based on signals of cardholder participation, not solely on API parameters” [25]. Classification runs on signals of cardholder participation, which a single API parameter does not settle [25], so an API parameter alone does not decide how an agent-initiated purchase is classified. **Where it is so classified on Visa, this paper’s own worked example exceeds the window.** In Appendix B.1 the token expires at 2026-10-07T12:20:00Z and the task’s deadline is 2026-10-12T17:00:00Z, so delivery may fall 5 days 4 hours 40 minutes after the latest activation, which is 10 hours 40 minutes beyond a Visa merchant-initiated window of 4 days 18 hours and inside the 7 days Stripe lists for Mastercard, American Express and Discover [25]. Extended authorisation reaches 29 days 18 hours on Visa for customer-initiated transactions only, subject to merchant category and account conditions [26]. A deployment therefore establishes the classification, network and window that apply to its own integration and uses the capture deadline the provider returns for each authorisation. Stripe’s stablecoin method supports no manual capture at all (§4.4). Capture remains subject to reversal, refund and provider-side dispute, which §14 places outside the model.

Mandate violations are refused with 403 at three points: on the frozen quote (§6 step 2), at `pending_authority` before any challenge is issued, and at the gate when the credential arrives; revocation is not among the conditions tested (§9.3). One found at the gate follows the transfer of §6 step 5 on a pre-funded method, and those funds are an unpaired provider effect, paired for a refund under O4. An incorrect or unnecessary purchase inside the mandate’s bounds can be admitted: the model does not establish the truth of a request or its usefulness, and the budget cap bounds what such purchases can spend across all of the mandate’s tasks (T5).

12 Sourcing and verification methodology

Every external factual claim about a protocol, specification, product, standard or regulatory instrument carries a citation to the References. Claims rest, in order of preference, on the protocol’s own specification or documentation, the originating organisation’s primary announcement, or the standards artefact itself; secondary reporting is used only where no primary source states a fact, and is tagged [S]. Where a reference carries an access date, that is the date recorded for reading it, and a fact cited from it is stated as of that date; the dates differ by entry and represent no single common verification date. An entry with no access date cites a fixed published version. Quoted text is verbatim, and a vendor’s description of its own protocol is attributed to that vendor. The agentic-payments research of §1.1 was located by searches of arXiv on 14 September 2026 for “agentic payments”, “agent payments protocol”, “HTTP 402” and “x402”; the works cited are those whose abstracts address admission, binding, replay, settlement or spend control.

Draft status. Claims drawn from the “Payment” HTTP authentication scheme are pinned to `draft-httpauth-payment-01`, dated 9 September 2026, an individual submission not endorsed by the IETF, document state Active, expiring 13 March 2027 [3]. The text cited is the one the IETF archive holds under that name; its SHA-256, taken from the archive on 21 September 2026 and unchanged on 27 September 2026, is

73cf80ac8fbad50c2eaa59dbac54b26352da3f79e5a235de408b42b5dbbe53db.

The same five authors earlier published `draft-ryan-httpauth-payment-01` under the same title. That document expired on 19 September 2026 and reached no -02, and the datatracker records no *replaces* relation in either direction, so the two are separate individual submissions, each at its own revision -01. The September text inserts Appendix B.2, *Challenge Negotiation with Accept-Payment*, and rennumbers every appendix after it, so App. B.3 in the March text is App. B.4 in the text this paper cites, and every section and appendix reference here is to the September text.

Tags. [S] marks a secondary source, [Draft] a live Internet-Draft, a draft in that format published by its authors outside the IETF datatracker, an expired draft cited for its text, or a feature its maintainer labels draft, experimental or preview, [Avail] availability that varies by country, entity, approval or waitlist, and [Vendor] a vendor’s framing of its own work.

Validation supplement. The validation supplement for this version, with its scripts under the MIT licence, is available from the author. The archive `mpp-validation-supplement-v2.9.zip` has SHA-256

844c1b342c0cba6a4e0752a68e9f53cc9463a5cedffedd12a84387d244bb55bf,

and any copy can be checked against it. It contains the checking scripts, the source files they parse with their SHA-256 hashes, the command and the expected output, and runs with Python 3 and no other dependency.

Method. Explicit-state enumeration in Python, breadth-first over a configuration tuple, with no external tool, no solver and no network. It checks the state, level and method sets of §9 as printed, named graph properties of the transition table of Appendix A, parsed as printed, the Appendix B example relations, including both limbs of the B.2 escalation predicate against a stated fixture, and, by their stated terms, the objects of C9 to C11 and Corollary 1 in the source text. It then explores *every* reachable configuration of five bounded instances, up to two mandates, three tasks and two admission attempts per task, and checks T1, T3, T5, Corollary 1 and the reservation-release property of O5 in each, that every entitlement admission draws on the record its token was minted against, and that a quoted task whose token has expired never takes its acceptance edge. The transition edges are parsed from the printed table, and a guard at quoted, pending_authority or the gate takes a class only when it is logically equivalent, over its atoms, to the guard the explorer implements for that class; the arithmetic guards and the spend and balance updates are encoded by hand in Python. The SOURCE checks tie that encoding to the printed displays textually: each display must carry exactly the conjuncts a table in the script lists, and each Python fragment the table names must occur in the explorer’s source, which the check does not execute. In the printed model every gate step is one atomic transition, so T1, T3, T5, the corollary and the release property hold by construction of the encoding; the run confirms that the encoding keeps them, and the altered models show that each counter fires. The bounded results therefore hold for that encoding, and passing does not establish that the whole printed model and the encoding are equivalent.

Size and runtime. The five instances have 58,695, 80,883, 1,705, 1,705 and 1,849 reachable configurations; the six altered models explore 186,461, 923,157, 194,565, 166,295, 296,277 and 145,829. Each altered-model figure is the total over the five instances. The third and fourth instances, two draws on one entitlement record under one mandate and under two, coincide because no cap binds in either; the fifth gives one client two entitlement records, with one task drawing on each. The whole supplement, all three scripts, runs in about 27 seconds on one core of an Apple-silicon laptop.

What must fail. Six altered models must each violate the property named for it, or the run exits non-zero: a budget tested before the gate with no reservation and no re-test at admission (T5); a gate that commits without re-checking the token (T3); a plan check that reads the entitlement balance at the gate and decrements it in a later step, which is the model without C11 (Corollary 1); a withdrawal at the gate that leaves the reservation standing (O5); an extra edge into Φ (T1); and a plan check without the membership conjunct of (13), which lets a request draw on another of the client’s records (the binding of the token to its record). The third, fourth and sixth leave every other property satisfied, which is why each is its own test, and a wider version of the first two would miss them. None of the six splits the budget test of C9 from its reservation; that atomicity rests on the constraint as stated. Thirty-two injected defects in copies of the parsed sources, or of the escalation fixture, must each fail the check named for them; Appendix D lists all six altered models and all thirty-two defects with the check each must fail. A check that no defect names has no shipped failing case.

What it does not check. The displayed proofs of T2 and T4, instances beyond its bounds, the equivalence of the printed model and its encoding beyond the SOURCE checks and the guard-equivalence check of the transition table, the prose of the three parsed files outside the displays, constraints, tables and listings the checks name, the paper’s prose outside those files, the PDF itself, which is compiled from those files and read by no script, and any implementation.

13 Assumptions

§13 states nine operating assumptions, each a condition that can be checked against a deployment. The table at its end shows which of them each result uses: six do work in a result, and A2, A4 and A5 scope the model and are used by no result.

A1 (Mandate and credential authenticity). A registered mandate and a presented payment credential are cryptographically verifiable, and TP can evaluate $\text{authentic}(\text{cred}, R)$ without trusting the calling agent. A mandate’s signature is verified when it is registered, and the registry entry is immutable thereafter; at activation the operator relies on that entry and on $\text{authn}(\text{holder}(M))$, and no step of §9.5 verifies a signature carried in the request. AP2-style mandates are taken to be tamper-proof, signed contracts in the sense Google describes [34], and the model relies on signature verification and transport authentication of the caller, never on the agent’s own assertion of authority. The mandate’s status (neither revoked nor suspended, issuer key valid) holds for its lifetime; revocation before expiry sits outside the enumerated failure path of §11.

A2 (Deterministic pricing). For a given product and parameter set, price returns a single value, frozen at quote time and honoured until the quote’s token expires, so two callers presenting the same parameters are quoted the same price.

A3 (Gate-before-fulfilment is enforceable). TP controls the ordering of its own task state machine, so no transition into Φ can occur before the gate is satisfied, under every admission method and every revision of an external protocol.

A4 (Admission-method availability varies). No payment method is assumed universally available. Availability varies by country, legal entity, account approval and currency (§4.4); at least one method is available for any transaction the model admits. Invoice and the three entitlements depend on no provider’s machine-payment acceptance, so a deployment with no external payment method may transact on billing and entitlements alone.

A5 (Expert supply). Enough qualified experts exist to satisfy a task’s χ_M within its quote’s deadline. Where supply fails, the task ends at `compensation_recorded`.

A6 (Hash and signature integrity). H is collision-resistant, so artefactHash binds one artefact and $\text{ph}(q)$ one parameter record. The signature scheme behind σ_Ω and the unlock grant is unforgeable, and the operator’s signing key is not compromised.

A7 (Stable client idempotency keys). A client generates a stable key per logical request and reuses it on retry, so T2 holds under network retry and at-least-once delivery. T3 does not depend on this assumption.

A8 (Receipt and ledger integrity). L is append-only, and the receipt store, task records, admission records and mandate registry are durable, so checks (2) to (5) of $\text{verify}\Omega$ read what was written.

A9 (Single-use token integrity). Nonces and drawdown tokens are minted by TP’s own engine, unpredictable and collision-free, and each consumption is atomic with the transition it guards, so a consumed token never becomes fresh again. The atomicity is over TP’s own stores, under the serialisations of **C9**, **C11** and §9.5; the payment provider lies outside it.

Result	Assumptions it uses	Constraints it uses
T1 gate-before-fulfilment	A3, A9	C1, C7, C10
T2 idempotency	A1, A7, A6, A9	C1, C8; T3
T3 single consumption	A9	C1, C7, C10
T4 auditability	A6, A8	T3
T5 budget-cap soundness	A9	C5, C9; T3

14 Scope & boundary conditions

In scope. The layered architecture (§§3 to 5), the transaction model and endpoint contracts (§§6 and 7), the formal model and its five results (§9, Appendix A), the implementation constraints and obligations (§10) and the business-model transition (§8).

Out of scope. No reference implementation is evaluated, and no adoption, market-size or revenue forecast is made. The paper cites MPP, the “Payment” scheme, MCP, AP2, A2A, UCP, ACP and x402 as their maintainers define them and does not specify their wire formats, the credential shape of any method, or an adapter’s conformance to a draft revision (O7). It does not model the payment provider’s lifecycle (authorisation, capture, settlement finality, reversal, chargeback), reconciliation against it, crash recovery or store-level transaction mechanics, and none of its results concerns the collection of funds. It does not specify the pricing function beyond determinism, expert sourcing and payout beyond their interface to the task engine and ledger, the security of cryptographic primitives, transport bindings beyond §6, adversarial expert content, tenant isolation, data classification, retention, or the operational layer of monitors, service-level objectives and recovery procedures.

Mainstream-user constraint. The architecture assumes non-crypto clients and experts and never forces crypto adoption on any participant; stablecoin is one supported method among eight and never the default.

Compliance and operating controls. The controls an expert-access practice already runs stay in place: expert vetting and credentialing, conflict-of-interest and independence screening, confidentiality handling, restricted-topic and material-non-public-information policy, client approval rules and recordkeeping. Two have counterparts in the model: expert vetting resolves through χ_M , a ledger invariant, and client approval through ε_M at `pending_authority`. The rest are deployment policy, run at four control points: request eligibility before the quote; screening of the selected expert in `pending_expert_match`; screening of each input for confidentiality and material non-public information in `awaiting_expert_input`; and release review in `in_review`, where a hold that ends in non-release exits by the unrecoverable-defect edge. The deployment’s compliance function owns each control point and records each decision. Whether that set suffices for a surface on which a mandate can admit a request without a per-request human decision is a determination this paper does not make.

Method selection. The choice of method within a branch is free at the gate and constrained outside it: merchant and acquirer roles, customer authentication, dispute and chargeback exposure, custody, sanctions screening, tax and accounting treatment and settlement finality differ by method and jurisdiction. Which role a deployment occupies in the funds flow, and the regulatory perimeter that follows, are determinations for the deployment (O6).

Manual exception path. Bespoke work that cannot be expressed as a fixed-price, bounded product stays on a human-mediated path (custom scoping, a quoted engagement or a live call). A2 fails for such work: where scope is settled in the conversation that scopes it, no deterministic price exists and no quote can be frozen.

15 Conclusion

For a purchasing agent under a bounded mandate, the architecture puts fixed-price expert products behind one gate state that separates quotation and authority from fulfilment, and admits a quote against a fresh payment or an entitlement the client already holds. The formal model establishes T1 to T4 by construction and proves T5 by induction, with a corollary bounding entitlement balances, under A1 to A9 and C1 to C11; the wire contract of §6 fixes how MPP challenges, credentials, refusals and escalations meet that gate; and obligations O1 to O8 carry what the model leaves to the provider and the deployment. How the model departs from the author's Internet-Draft is set out in §1.1. Admission records a credential, a receivable or a consumed allowance, and the limits of these results are those of §14.

Declarations

Competing interests. The author is the founder of True Primary, the firm whose practice and target architecture this paper describes. The paper makes no claim of deployed product capability.

Funding. This research received no external funding.

Code availability. A validation supplement was built for this version and is available from the author; §12 gives its archive hash and what it checks.

Companion note. *A Payment and Entitlement Gate for Self-Serve Agentic Expert-Access: A Practitioner Note*, version 1.9, written for implementers, is available from the author.

Data availability. No empirical dataset was generated or analysed, and the paper reports no observed prices: the amounts in §6 and Appendix B are illustrative. §8 is commercial context and carries no demand, margin or conversion-rate evidence.

AI use. The author conceived this paper and its argument, directed its development, verified its content, and takes full responsibility for it. AI tools assisted with drafting, editing, code and typesetting. AI was not used to generate scientific content, analyses, interpretations, or conclusions.

Appendix A: Task state machine (full transition table)

State	Meaning
draft	request captured under its mandate; not yet priced
quoted	price, scope, deadline and admission token issued (R on the payment branch, z on the entitlement branch)
accepted	the client has accepted the quote
pending_authority	authority and escalation resolved here; no fulfilment cost
pending_payment_or_plan_check	the gate g^* : admission by validate ($S \in Chal$) or plancheck ($S \in Ent$)
pending_expert_match	(Φ) sourcing experts satisfying χ_M
awaiting_expert_input	(Φ) experts contributing micro-inputs to L
needs_client_clarification	(Φ) blocked on a clarifying answer during fulfilment
in_review	(Φ) aggregation, validation, quality review and release review
ready	(Φ) artefact generated; Ω and σ_Ω being constructed
delivered	(Φ) artefact and signed receipt returned
cancelled	ended before the gate edge; no expert cost incurred
failed	unrecoverable system fault; after the gate a compensating record is written
compensation_recorded	post-gate compensating record written against ρ_{pay}

Guards are necessary preconditions, evaluated on the in-force (M, q) with $M = md(t)$, and time-driven guards are evaluated before client events. The per-state guards are mutually exclusive for each state; the cross-cutting fault guard of the closing row is not, and the model does not order the two. $\omega = tok(q)$. A refused activation request at g^* matches no row and fires no transition (§9.4).

From	Guard	To
draft	submit	quoted
draft	abandon	cancelled
quoted	accept $\wedge \neg \text{expired}(\omega)$	accepted
quoted	reject $\vee \text{expired}(\omega)$	cancelled
accepted	begin authority check	pending_authority
pending_authority	expired(ω)	cancelled
pending_authority	$\neg \varepsilon \wedge \text{authorises}$	pending_payment_or_plan_check
pending_authority	$\neg \varepsilon \wedge \neg \text{authorises}$	cancelled
pending_authority	$\varepsilon \wedge \text{approved} \wedge \text{authorises}$	pending_payment_or_plan_check
pending_authority	$\varepsilon \wedge (\text{declined} \vee \neg \text{authorises})$	cancelled
pending_payment_or_plan_check	admit(q, S, M) = $\top$ (GATE)	pending_expert_match
pending_payment_or_plan_check	expired(ω) $\vee$ expired(V)	cancelled
pending_payment_or_plan_check	withdraw	cancelled
pending_expert_match	expert e with $\chi_M(e)$ found	awaiting_expert_input
pending_expert_match	no qualifying expert	compensation_recorded
awaiting_expert_input	input ℓ appended to L	awaiting_expert_input
awaiting_expert_input	clarification required	needs_client_clarification
awaiting_expert_input	inputs sufficient	in_review
needs_client_clarification	answer received	awaiting_expert_input
needs_client_clarification	timeout $\vee$ withdraw	compensation_recorded
in_review	review passed	ready
in_review	unrecoverable defect	compensation_recorded
ready	Ω, σ_Ω constructed $\wedge$ written	delivered
delivered	dispute upheld	compensation_recorded
any $s \in Q \setminus Term$	unrecoverable system fault	failed

The *declined* guard reads a recorded decline, and approved a recorded approval (§9.3). *expired(V)* applies on the entitlement branch, to the record the quote’s token was minted against. A *no qualifying expert* outcome is a post-gate business outcome and ends at `compensation_recorded`; *failed* is reserved for a system fault. A fault before the gate edge leaves `task(taskId)` undefined, so no token was consumed and nothing is compensated; a fault after it writes a compensating record against ρ_{pay} . Admission and compensation status are read off the task record, never off the terminal label. For an external-payment method, completion of a refund with the paying instrument is a provider outcome reconciled under O4. A release review at `in_review` that ends in non-release exits by the *unrecoverable defect* row.

Facts of this table used by the results of §9.8:

- The only edge from $Q \setminus \Phi$ into Φ is the gate edge, which both admitting paths guard (T1)
- `pending_authority` and the gate exit only to g^* , `pending_expert_match`, `cancelled` or `failed` (T1)
- `needs_client_clarification` is entered only from `awaiting_expert_input` (T1)
- The only in-edge to `delivered` writes Ω and σ_Ω , and no edge returns to `ready` (T4)
- Compensation consumes no token and writes a record against ρ_{pay} (T3)
- Time-driven exits exist at `quoted`, `pending_authority`, the gate and `needs_client_clarification` (O5)

Appendix B: Endpoint schemas

JSON and header shapes for one purchase. `payment_requirement` is R , quote carries fields of q , `mandate_id` names M in the operator’s registry, and the receipt object is Ω with σ_Ω . Identifiers, hashes, timestamps and amounts (minor units) are illustrative.

B.1 Quote with its requirement

The operator holds R and the quote; `offered_methods` is deployment data outside R . This GBP quote offers SPT and invoice. Stripe’s stablecoin method presents in USD (§4.4), so it is offered only for USD quotes. The quote response carries no challenge; the challenges answer the activation request (B.2). Invoice is named in `offered_methods` and answered with a billing approval off the wire (§9.4); it is not a registered MPP method, and the scheme’s method registry is Specification Required [3]. `payment_requirement` prints the fields of R the agent reads, and its `expires_at` is the expires every challenge for this quote carries:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Type: application/json
```

```
{
  "quote": {
    "quote_id": "q_8f3c0a2e",
    "agent": "agent:acme-buyer-01",
    "product": "self-serve/fixed-price-brief",
    "scope": "self-serve/fixed-price-brief",
    "description": "single-question expert brief",
    "params_hash": "sha256:1b9d...c7",
    "deadline": "2026-10-12T17:00:00Z"
  },
  "payment_requirement": {
    "amount": 50000, "currency": "GBP",
    "payee": "tp:merchant:trueprimary",
    "nonce": "n_4a7e91d2c5b08f63", "scheme": "Payment",
    "expires_at": "2026-10-07T12:20:00Z"
  },
  "offered_methods": ["spt", "invoice"]
}
```

B.2 The activation request and its 402

The activation request of step 4 of §6. Its body is the request of step 1 with `quote_id` and `params_hash` added. A client paying by SPT sends it twice with that body unchanged: first without a credential, which draws the 402 below, then with the selected credential under the same Idempotency-Key. A client paying this quote by invoice adds `"admission_method": "invoice"` to the body and sends it once, drawing 200 where a billing approval for *R* is recorded, or 202 with a status link while one is pending, and no challenge (§6 step 4). **Only the external-payment methods take a wire challenge.** The 402 carries one WWW-Authenticate: Payment challenge per offered method *in Ext* [3, App. B.4], each mapped by the adapter to the one nonce (O7), so this quote draws one challenge, and a payment-required problem body [3, §8], omitted here. Each challenge's digest parameter is the Content-Digest of the body that drew it, computed from the request in hand, and the retry satisfies it by carrying that body unchanged [3, §5.1.3]. The caller authenticates as the mandate holder with an HTTP message signature [5], whose covered components must be present in the message, so Content-Digest is sent as well as signed. **The covered components below are the minimum this example needs and are not a complete application profile.** RFC 9421 leaves the profile to the application, and this paper fixes none of it: a deployment states the authority the signature binds, by covering @authority or by confining acceptance to an authenticated operator registry; the algorithms and trusted keys it accepts; its handling of created and expires and the clock skew it tolerates; its replay treatment; and whether mutual TLS is accepted in place of a signature (§6). It tests a wrong-authority, an expired and a modified-body request against each. The body carries `mandate_id`; it omits the full signed mandate object, which the operator reads from its registry. The draft's grammar puts the base64url credential directly after the scheme name [3].

```
POST /self-serve/fixed-price-brief HTTP/1.1
Idempotency-Key: "ik_2f9c7b13e0a4"
Signature-Input: sig1=("@method" "@path" "idempotency-key"
  "content-digest");keyid="agent:acme-buyer-01";created=...
Signature: sig1:...:
Content-Digest: sha-256=...:
```

```
Content-Type: application/json
```

```
{
  "quote_id": "q_8f3c0a2e",
  "params_hash": "sha256:1b9d...c7",
  "mandate_id": "m_c19a44",
  "request": {
    "question": "Demand outlook for advanced-node foundry capacity, H2 2026.",
    "params": { "depth": "standard", "format": "memo" }
  }
}
```

```
HTTP/1.1 402 Payment Required
Cache-Control: no-store
WWW-Authenticate: Payment id="...", realm="...", method="stripe",
  intent="charge", expires="2026-10-07T12:20:00Z", request="...",
  digest="sha-256:...:"
```

The wire method is the provider's registered name. The internal enumeration of §9.4 labels this admission method `spt`, and it maps to the wire method `stripe` under the `charge` intent, with the shared payment token carried inside the credential at `payload.spt` [8]. The internal label appears in `admission.mode` and never on the wire.

The retry, with the body above unchanged:

```

POST /self-serve/fixed-price-brief HTTP/1.1
Authorization: Payment <base64url-encoded credential>
Idempotency-Key: "ik_2f9c7b13e0a4"
Signature-Input: sig1=( "@method" "@path" "idempotency-key"
    "content-digest");keyid="agent:acme-buyer-01";created=...
Signature: sig1=:...:
Content-Digest: sha-256=:...:
Content-Type: application/json

```

The registry entry for this mandate reads as follows.

```

mandate_id m_c19a44
holder agent:acme-buyer-01
budget_cap 100000 GBP
authorised_scope self-serve/fixed-price-brief, self-serve/quick-validation
authority_expires 2026-10-30T00:00:00Z
expert_criteria sector=semiconductors;min_years=10
escalation_if amount_minor > 75000
                or topic in client_restricted_list
                    (evaluated on the quote)

```

With nothing else committed or reserved, the quote meets all seven conjuncts of (4): the signature authenticates the holder; the holder is the quote's agent; the scope is authorised; the currencies match; $0 + 50000 \leq 100000$; the deadline precedes the authority expiry; and the request arrives before it. Escalation is false on both limbs: the amount is 50000, at or below the 75000 threshold, and the quote's topic is absent from `client_restricted_list`. Either limb alone would make $\varepsilon_M(q)$ true, so a worked decision states both; a quote whose topic is on that list escalates at any amount. The gate recomputes the parameter digest over request and admits only where it equals the quote's `params_hash` (§9.5); that check is the price-freeze binding of C1, and the body digest the challenge binds is a separate one. The same request against a quote scoped `self-serve/market-pulse` fails scope membership and is answered 403.

B.3 Responses to the activation request

Admitted (steps 6 and 7 of §6). Any Payment-Receipt header is the selected method's wire payment receipt [12], on a response marked `Cache-Control: private`.

```

{
  "status": 200,
  "task": { "task_id": "t_5d2b8e1f", "state": "pending_expert_match",
            "quote_id": "q_8f3c0a2e",
            "result_endpoint": "/paid-artifacts/a_7c40e9b2" },
  "admission_disposition": "admitted",
  "idempotent_replay": false
}

```

Four conditions answer 202, at two states. **Escalation pending** holds the task at `pending_authority` awaiting the recorded human decision (C6):

```
{
  "status": 202,
  "task": { "task_id": "t_5d2b8e1f", "state": "pending_authority" },
  "admission_disposition": "pending",
  "status_endpoint": "/quotes/q_8f3c0a2e"
}
```

At the gate, an indeterminate provider outcome, a customer action the provider requires and an invoice approval still pending each leave the task at the gate with the disposition stored (O3), a different state with a different remedy; a customer-action response also carries the provider's action reference beside the status link. The example is the indeterminate case:

```
{
  "status": 202,
  "task": { "task_id": "t_5d2b8e1f",
            "state": "pending_payment_or_plan_check" },
  "admission_disposition": "pending",
  "status_endpoint": "/quotes/q_8f3c0a2e"
}
```

A refused credential answers 402 with fresh challenges and a problem body [3, §8.2, App. B.5], after which the task stays at the gate:

```
{
  "type": "https://paymentauth.org/problems/verification-failed",
  "title": "Payment Verification Failed",
  "status": 402,
  "task_state": "pending_payment_or_plan_check"
}
```

B.4 The signed delivery receipt

Written once at the delivery transition (§9.6). A verifier resolves the key named by `key_id` within its configured trusted key set for this operator, checks `signature` under that key, then runs the five checks, reading `artefactHash` from `artefact_hash`, ρ_{pay} from `payment_ref`, M from `mandate_id`, $m(q_t)$ from `amount` and `currency`, t_{del} from `delivered_at`, and `mode` from `admission.mode`. **All seven fields of Ω appear in the object and inside $\text{canon}(\Omega)$** , so the printed receipt instantiates the signed tuple of (18) exactly, and a receipt whose `delivered_at` is altered fails verification as any other signed field does. On the card path under the pinned intent the credential was charged at admission, with no capture at delivery (§11). **Five fields lie outside $\text{canon}(\Omega)$ and are unsigned:** `receipt_id`, `key_id`, `issued_at`, `admission.instrument` and `admission.via_mpp`. `key_id` is a lookup hint carrying no authority of its own: it selects a key from the verifier's trusted set for this operator, so a changed `key_id` selects a different trusted key or none. `receipt_id` and `admission.instrument` are transport and display aids no check reads, and `issued_at` is the operator's write time, a different quantity from the signed t_{del} , and is not verified. `admission.via_mpp` is derivable from `admission.mode`.

The canonical form and the signature profile are a deployment obligation this paper does not discharge. `canon` is required to be injective (§9.2); nothing here fixes a byte encoding, a signature algorithm, a schema version or a test vector. A deployment publishes all four, with its key rotation and retention rules, and ships a positive and a negative vector for each signed field, before a third party can verify a receipt it did not receive from the operator.

```
{
  "receipt_id": "rcpt_a0f4c2", "task_id": "t_5d2b8e1f",
  "mandate_id": "m_c19a44", "payment_ref": "pay_7c3e90",
  "artefact_hash": "sha256:4e2a...b18f",
  "amount": 50000, "currency": "GBP",
  "delivered_at": "2026-10-09T15:42:05Z",
  "issued_at": "2026-10-09T15:42:07Z",
  "admission": { "mode": "spt", "instrument": "card", "via_mpp": true },
  "key_id": "tp-receipts-2026-09",
  "signature": "<detached signature over the canonical receipt>"
}
```

B.5 The entitlement branch

A client on a package asks for the same brief. The quote carries a drawdown token and no challenge; the activation request carries no Authorization header and is admitted on its first arrival; the admission records the entitlement the token names, the one record (13) lets it draw on.

```
HTTP/1.1 200 OK
{ "quote": { "quote_id": "q_91d0b7c4", "...": "as B.1" },
  "drawdown_token": { "token_id": "z_6b21e0",
    "entitlement": "package_drawdown", "amount": 50000,
    "currency": "GBP", "expires_at": "2026-10-07T12:20:00Z" } }

POST /self-serve/fixed-price-brief HTTP/1.1
Idempotency-Key: "ik_8d33a6f1c2b9"
Signature-Input: ...
Signature: ...
{ "quote_id": "q_91d0b7c4", "params_hash": "sha256:1b9d...c7",
  "mandate_id": "m_c19a44", "request": { "...": "as B.2" } }

HTTP/1.1 200 OK
{ "task": { "task_id": "t_0c77e2a9", "state": "pending_expert_match" },
  "admission_disposition": "admitted",
  "admission": { "mode": "package_drawdown", "via_mpp": false } }
```

Appendix C: Protocol status and sources

Status as of the access date of each cited reference. Secondary reports are marked [S], drafts and preview features [Draft], availability constraints [Avail] and vendor framing [Vendor].

Protocol	Originator	Primitive and role here	Governance	Version and stability	True Primary's decision
MPP	Tempo Labs and Stripe; Visa publishes a card specification [6, 11]	402 with the “Payment” authentication scheme; wire protocol for the payment branch [7, 3]	Specification maintained by Tempo Labs and Stripe [6]	Launched 18 March 2026 [10]; eleven methods and a custom option listed [8]; core scheme an individual Internet-Draft, pinned here at draft-httpauth-payment-01, active, expiring 13 March 2027 [3] [Draft], while the repository links draft-ryan-httpauth-payment [6]; Stripe's integration is in preview (§4.4) and its MCP guide labelled experimental [22, 21, 14] [Draft]; availability per §4.4 [Avail]	Candidate wire protocol (core)
AP2	Google [34]	Signed mandates; reference model for <i>M</i> [35]	Contributed to the FIDO Alliance, announced 28 April 2026 [36]; standardisation continues in FIDO's Agentic Authentication Technical and Payments Technical Working Groups [35]	v0.2, with Checkout and Payment Mandates [35]	Core for delegated authority
MCP	Anthropic, 25 November 2024 [32]	Tools, resources and prompts for AI applications; access layer [16]	Agentic AI Foundation, a directed fund under the Linux Foundation, since 9 December 2025 [33]	Protocol version 2026-07-28 [30]; that revision made the protocol stateless and moved Tasks into the extension io.modelcontextprotocol/tasks [31]	Core for access; Tasks as interface mapping
A2A	Google, donated to the Linux Foundation [38]	Agent-to-agent communication [37]	Linux Foundation project formed 23 June 2025 [38]; the project site announces that A2A joins the Agentic AI Foundation [37]	v1.0 protocol definition [37]	Conditional, for a coordinating agent
UCP	Google and partners [41]	Commerce-journey abstraction [Vendor] [41]; channel wrapper	Maintained through its protocol site [39]	Date-based versions, specification 2026-08-25 [39]; merchant participation on Google by waitlist [40] [Avail]	Conditional wrapper
ACP	Stripe and OpenAI [42]	Checkout under a merchant-of-record model; channel wrapper [42]	Open source under Apache 2.0 [42]	Launched 29 September 2025 [42]; purchases of Shopify merchants' products found in ChatGPT reported moving to the merchant's own store, March 2026 [43] [S]	Conditional wrapper
x402	Coinbase; x402 Foundation [44]	402 with its own headers [45]; historical paid-resource pilot	Linux Foundation announced the launch of the x402 Foundation with Coinbase's contribution, 2 April 2026 [44]	Protocol version 2 HTTP transport [45]	Not the client-facing default

Appendix D: What the validation supplement must fail

A check that cannot fail is not evidence. The supplement of §12 therefore ships its own falsification set: six altered models, each of which must violate at least the property named for it, and thirty-two injected defects, each of which must fail the named check. The run exits non-zero if any of them passes. The scripts print these counts to standard output, and the tables below are generated from the run’s recorded output, `EXPECTED-OUTPUT.txt`. The harness is a bounded regression check over an encoded finite model: it does not instantiate signatures, provider calls, revocation, the paid unlock, or agreement with the Internet-Draft or the note.

D.1 Altered models. Configuration counts are from the run whose output the archive ships as `EXPECTED-OUTPUT.txt`.

Altered model	What is changed	Must violate	Configs
NO-RESERVATION	the budget is tested before the gate against committed spend only, with no reservation and no re-test at admission	T5	186,461
UNSERIALISED	the gate evaluates and commits in separate steps without re-checking the token, so two attempts on one task can both commit	T3	923,157
UNSERIALISED-DRAW	the plan check reads $\text{bal}(V)$ at the gate and decrements it in a later step, which is the model without C11	Corollary 1	194,565
WITHDRAW-LEAKS	the client’s withdrawal at the gate leaves the reservation standing, so the gate has no client-driven exit that releases it	O5 release	166,295
EXTRA-EDGE	Appendix A with one added edge $\text{pending_authority} \rightarrow \text{pending_expert_match}$	T1	296,277
UNBOUND-DRAW	the plan check drops its membership conjunct $\text{draw}(q) \in Z_{\text{ent}}(V)$, so a request may draw on another of the client’s records	token-record binding	145,829

The third, fourth and sixth leave every other checked property satisfied (§12), which is why each is listed separately: Corollary 1, the release property of O5 and the token-record binding are the only checks that see them.

D.2 Injected defects. Each is one textual change applied to a copy of the parsed sources or of the escalation fixture.

Injected defect	Check that must fail
refusal row added at the gate	FSM/no-refusal-row-at-gate
second edge into the fulfilment region	FSM/only-edge-into-Phi-is-gate-edge
expiry exit removed at pending_authority	FSM/time-driven-exits-at-four-states
delivery edge no longer writes the receipt	FSM/delivered-only-from-ready-writing-receipt
out-edge added to an exception terminal	FSM/exception-terminals-have-no-out-edges
quoted acceptance guard loses its expiry conjunct	FSM/quoted-acceptance-guard-is-accept-and-not-expired
escalated approval made vacuous by OR true	FSM/guarded-rows-equivalent-to-the-explored-guards
admission guard made vacuous by OR true	FSM/guarded-rows-equivalent-to-the-explored-guards
gate moved to level 5	SETS/gate-at-level-4
stablecoin dropped from Chal	SETS/Chal-is-Ext-disjoint-union-Bill
Eq. (13) balance guard replaced by a true conjunct	SOURCE/planchek-balance-guard-printed
drawdown token no longer bound to the drawn record	SOURCE/planchek-token-bound-to-record-guard-printed
admission predicate made unconditional	SOURCE/admit-is-validate-or-planchek
stablecoin offered for a GBP quote	EXAMPLE/no-method-offered-outside-its-presentment-currency
cap below the price	EXAMPLE/authority-budget-0-plus-price-within-cap
cap in another currency	EXAMPLE/authority-currency-matches
escalation threshold below the price	EXAMPLE/escalation-predicate-false-for-quote
escalation amount limb made always true	EXAMPLE/escalation-predicate-false-for-quote
fixture topic placed on the restricted list	EXAMPLE/escalation-predicate-false-for-quote
receipt signature emptied	EXAMPLE/receipt-carries-signature-and-key
refused credential shown leaving the gate	EXAMPLE/refused-credential-leaves-task-at-gate
receipt method not offered	EXAMPLE/receipt-mode-is-an-offered-admission-method
payment credential on the entitlement branch	EXAMPLE/entitlement-branch-has-no-payment-credential
Idempotency-Key value printed bare	EXAMPLE/idempotency-key-values-are-sf-strings
pending status link keyed by the task	EXAMPLE/pending-status-endpoint-is-keyed-by-quote
entitlement admission recorded on another record	EXAMPLE/entitlement-admission-draws-on-the-token-record
C9 weakened to separate test and reservation steps	CONSTRAINT/C9-tests-and-reserves-in-one-atomic-step
C11 removed, so no per-entitlement serialisation is stated	CONSTRAINT/C11-serialised-entitlement-draw-exists
C11 weakened to separate read and decrement steps	CONSTRAINT/C11-orders-the-two-reads-in-one-atomic-step
balance corollary removed	CONSTRAINT/balance-corollary-exists
balance proof rests on the token instead of the serialisation	CONSTRAINT/balance-proof-rests-on-C11
C10 reverted to promising a reservation	CONSTRAINT/C10-names-a-token-not-a-reservation

The CONSTRAINT rows remove or weaken the printed text of the concurrency constraints; against a copy of the sources with that text taken out, the CONSTRAINT checks those rows name fail. Those checks read the terms each constraint states in the source text; they do not execute it, which the state-space run does over its hand-encoded guards.

References

- [1] K. Y. Choo. A Server-Side Model for Gating Agent-Initiated Human-Sourced Asynchronous HTTP Tasks on Payment or Entitlement. Internet-Draft draft-king-yew-choo-agentic-payments-00, individual submission, Informational, 23 July 2026, expires 24 January 2027. <https://datatracker.ietf.org/doc/draft-king-yew-choo-agentic-payments/00/>, accessed 14 September 2026, [Draft]. **On 29 September 2026 revision -01 had been written and was not yet filed**, so -00 was the revision on the record and is what this paper cites; the differences §1.1 records are against -00.
- [2] I. Schrock (EMILIA Protocol, Inc.). The Action Evidence Boundary for Consequential Agent Effects. Internet-Draft draft-schrock-action-evidence-boundary-05, Informational, 31 August 2026. <https://datatracker.ietf.org/doc/draft-schrock-action-evidence-boundary/05/>, accessed 27 September 2026, when the datatracker listed revisions to -07 and the draft's state as Active, [Draft]. The sentences quoted in §1.1 read the same in revision -00 (21 July 2026) and in -05.
- [3] B. Ryan, J. Moxey and T. Meagher (Tempo Labs), J. Weinstein and S. Kaliski (Stripe). The “Payment” HTTP Authentication Scheme. Internet-Draft draft-httpauth-payment-01, individual submission not endorsed by the IETF, updated 9 September 2026, expires 13 March 2027, document state Active. <https://datatracker.ietf.org/doc/draft-httpauth-payment/>; text <https://www.ietf.org/archive/id/draft-httpauth-payment-01.txt>; accessed 27 September 2026, [Draft]. The same five authors earlier published draft-ryan-httpauth-payment-01 under the same title, a different text: the September text runs 2,016 lines against 1,624 and inserts an appendix (§12).
- [4] J. Jena and S. Dalal. The Idempotency-Key HTTP Header Field. Internet-Draft draft-ietf-httpapi-idempotency-key-header-07, HTTPAPI working group, 15 October 2025, expired 18 April 2026. <https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/>, accessed 29 September 2026, [Draft].
- [5] A. Backman, J. Richer and M. Sporny. HTTP Message Signatures. RFC 9421, February 2024. <https://www.rfc-editor.org/rfc/rfc9421>.
- [6] Machine Payments Protocol specification (tempoxyz/mpp-specs). <https://github.com/tempoxyz/mpp-specs>, accessed 27 September 2026, primary (specification repository).
- [7] Machine Payments Protocol: Overview. <https://mpp.dev/overview>, accessed 14 September 2026, primary (protocol site).
- [8] Machine Payments Protocol: Payment methods. <https://mpp.dev/payment-methods/>, accessed 27 September 2026; the Stripe charge method and its credential payload, <https://mpp.dev/payment-methods/stripe/charge>, accessed 28 September 2026, primary (protocol site).
- [9] Introducing the Machine Payments Protocol, Stripe blog, 18 March 2026. <https://stripe.com/blog/machine-payments-protocol>, accessed 14 September 2026, primary (vendor announcement).
- [10] Tempo Mainnet is live, Tempo Labs blog, 18 March 2026. <https://tempo.xyz/blog/mainnet/>, accessed 14 September 2026, primary (vendor announcement).

- [11] Visa card specification and SDK for the Machine Payments Protocol, Visa, 18 March 2026. <https://corporate.visa.com/en/sites/visa-perspectives/innovation/visa-card-specification-sdk-for-machine-payments-protocol.html>, accessed 14 September 2026, primary (vendor announcement).
- [12] J. Brans (Visa). Card Network Charge Intent for HTTP Payment Authentication. Draft in Internet-Draft format, draft-card-charge-00, published at <https://paymentauth.org/draft-card-charge-00> on 9 September 2026 and republished under the same name on 26 September 2026, expires 30 March 2027; absent from the IETF datatracker as of 28 September 2026, where a search for card-charge returns no document. Accessed 28 September 2026, [Draft].
- [13] Machine payments, Stripe documentation. <https://docs.stripe.com/payments/machine>, accessed 20 September 2026, primary (vendor documentation).
- [14] Monetize an MCP server, Stripe documentation. <https://docs.stripe.com/agent-commerce/monetize-mcp>, accessed 20 September 2026, primary (vendor documentation), [Draft].
- [15] MCP SDK, Machine Payments Protocol partner integrations. <https://mpp.dev/partner-integrations/mcp-sdk>, accessed 20 September 2026, primary (protocol site).
- [16] Model Context Protocol specification, revision 2026-07-28. <https://modelcontextprotocol.io/specification/2026-07-28>, accessed 20 September 2026, primary (specification).
- [17] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 249–259, 1987. DOI 10.1145/38713.38742.
- [18] G. Pardon and C. Pautasso. Towards Distributed Atomic Transactions over RESTful Services. In *REST: From Research to Practice*, pages 507–524. Springer, New York, 2011. DOI 10.1007/978-1-4419-8303-9_23.
- [19] A. Birgisson, J. G. Politz, Ú. Erlingsson, A. Taly, M. Vrabie and M. Lentczner. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Network and Distributed System Security Symposium (NDSS)*, 2014.
- [20] J. Richer (Ed.) and F. Imbault. Grant Negotiation and Authorization Protocol (GNAP). RFC 9635, October 2024. <https://www.rfc-editor.org/rfc/rfc9635>.
- [21] MPP, Stripe documentation. <https://docs.stripe.com/payments/machine/mpp>, accessed 14 September 2026, primary (vendor documentation).
- [22] Shared payment tokens, Stripe documentation. <https://docs.stripe.com/agent-commerce/concepts/shared-payment-tokens>, accessed 14 September 2026; its seller terms, Stripe Agent Commerce Seller Services (Preview), last modified 28 September 2026, <https://stripe.com/legal/ssa-services-terms>, accessed 29 September 2026, primary (vendor documentation), [Draft] (preview terms).
- [23] Stablecoin payments, Stripe documentation. <https://docs.stripe.com/payments/stablecoin-payments>, accessed 14 September 2026, primary (vendor documentation), [Avail].
- [24] Stablecoin Payments Terms (United Kingdom), Stripe. <https://stripe.com/gb/legal/stablecoin-payments>, last updated 16 April 2026, accessed 29 September 2026, primary (published contract terms). The definitions of the settlement services and the risk, liability and depeg allocations cited in §4.4 are in this document and are not in the documentation page above.
- [25] Place a hold on a payment method, Stripe documentation. <https://docs.stripe.com/payments/place-a-hold-on-a-payment-method>, accessed 20 September 2026, primary (vendor documentation).
- [26] Place an extended hold on an online card payment, Stripe documentation. <https://docs.stripe.com/payments/extended-authorization>, accessed 20 September 2026, primary (vendor documentation).

- [27] A new regime for cryptoasset regulation, Financial Conduct Authority. <https://www.fca.org.uk/firms/new-regime-cryptoasset-regulation>, accessed 14 September 2026, primary (regulator).
- [28] Guiding and Establishing National Innovation for U.S. Stablecoins Act, Public Law 119-27, 18 July 2025. <https://www.govinfo.gov/app/details/PLAW-119publ27>, accessed 14 September 2026, primary (statute).
- [29] Model Context Protocol: Tools, revision 2026-07-28. <https://modelcontextprotocol.io/specification/2026-07-28/server/tools>, primary (specification).
- [30] Model Context Protocol: Versioning. <https://modelcontextprotocol.io/specification/versioning>, accessed 14 September 2026, primary (specification).
- [31] Model Context Protocol: Key changes, revision 2026-07-28. <https://modelcontextprotocol.io/specification/2026-07-28/changelog>, accessed 14 September 2026, primary (specification).
- [32] Introducing the Model Context Protocol, Anthropic, 25 November 2024. <https://www.anthropic.com/news/model-context-protocol>, accessed 14 September 2026, primary (announcement).
- [33] Donating the Model Context Protocol and establishing the Agentic AI Foundation, Anthropic, 9 December 2025. <https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of-the-agentic-ai-foundation>, accessed 14 September 2026, primary (announcement).
- [34] Announcing Agent Payments Protocol (AP2), Google Cloud blog, 16 September 2025. <https://cloud.google.com/blog/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol>, primary (vendor announcement).
- [35] Agent Payments Protocol. <https://ap2-protocol.org/>, accessed 27 September 2026; the specification page, headed “Agentic Payment Protocol (v0.2)”, <https://ap2-protocol.org/ap2/specification/>, accessed 28 September 2026, primary (specification).
- [36] FIDO Alliance to develop standards for trusted AI agent interactions, FIDO Alliance, 28 April 2026. <https://fidoalliance.org/fido-alliance-to-develop-standards-for-trusted-ai-agent-interactions/>, accessed 14 September 2026, primary (announcement).
- [37] Agent2Agent (A2A) protocol. <https://a2a-protocol.org/latest/>, v1.0 protocol definition, accessed 27 September 2026, primary (specification).
- [38] Google Cloud donates A2A to Linux Foundation. <https://developers.googleblog.com/en/google-cloud-donates-a2a-to-linux-foundation/>, project formed 23 June 2025, primary (announcement).
- [39] Universal Commerce Protocol. <https://ucp.dev/>; specification 2026-08-25, <https://ucp.dev/2026-08-25/specification/overview/>, which is what the undated <https://ucp.dev/specification/overview/> serves; accessed 21 September 2026, primary (protocol site).
- [40] Universal Commerce Protocol, Google for Developers. <https://developers.google.com/merchant/ucp>, accessed 14 September 2026, primary (vendor documentation), [Avail].
- [41] Under the Hood: Universal Commerce Protocol (UCP), Google Developers blog, 11 January 2026. <https://developers.googleblog.com/en/under-the-hood-universal-commerce-protocol-ucp/>, primary (vendor announcement).
- [42] Developing an open standard for agentic commerce, Stripe blog, 29 September 2025. <https://stripe.com/blog/developing-an-open-standard-for-agentic-commerce>, accessed 14 September 2026, primary (vendor announcement).

- [43] “Shopify changes its integration approach with ChatGPT”, Digital Commerce 360, 17 March 2026. <https://www.digitalcommerce360.com/2026/03/17/shopify-integration-with-chatgpt-changes/>, secondary [S].
- [44] Linux Foundation is launching the x402 Foundation, The Linux Foundation, 2 April 2026. <https://www.linuxfoundation.org/press/linux-foundation-is-launching-the-x402-foundation-and-welcoming-the-contribution-of-the-x402-protocol>, accessed 14 September 2026, primary (announcement).
- [45] x402 v2 HTTP transport specification. <https://github.com/coinbase/x402/blob/main/specs/transport-s-v2/http.md>, accessed 14 September 2026, primary (specification).
- [46] K. Jiang, M. Yu, Y. Chang, M. K. Jangid, J. Niu, C. Wang and Y. Zhang. A Formal Analysis of Agent Payment Protocols. arXiv:2609.00060v1, August 2026.
- [47] S. Ling, Y. Huang, Y. Du, Y. Chen, Y. Zhou, L. Wu and C. Wang. Free-Riding the Agentic Web: A Systematic Security Analysis of x402 Payments. arXiv:2605.30998v2, June 2026.
- [48] Y. Li, L. Wang, K. Wang, Z. Yang, K. Wang, Z. Guan and J. Gao. A402: Binding Cryptocurrency Payments to Service Execution for Agentic Commerce. arXiv:2603.01179v2, March 2026.
- [49] B. Mohammadkhani, A. Khekade and R. Kakkad. Agentic Settlement Protocol: An Application Profile for Refundable, Delayed-Fulfilment Agent Commerce on Stablecoin Rails. arXiv:2609.02208v1, September 2026.
- [50] M. S. Uddin, M. Mouzam, M. Imran and S. B. U. Faizan. APEX: Agent Payment Execution with Policy for Autonomous Agent API Access. arXiv:2604.02023v1, April 2026.
- [51] Q. Lan, A. Kaul, S. Jones and S. Westrum. Zero-Trust Runtime Verification for Agentic Payment Protocols: Mitigating Replay and Context-Binding Failures in AP2. arXiv:2602.06345v1, February 2026.
- [52] A. Aviv, P. A. Gandh, R. Bitton and A. Shabtai. Beyond the Mandate: A Systematic Security Analysis of the Agent Payments Protocol (AP2). arXiv:2608.23858v1, August 2026.
- [53] Y. Zhang, Y. Xiang, Y. Lei, Q. Wang, T. Qiu, Y. Sun, S. Zarkov, T. H. Yuen, A. Deppeler, J. Yu and K.-Y. Lam. SoK: Blockchain Agent-to-Agent Payments. arXiv:2604.03733v1, April 2026.
- [54] N. Asokan, M. Schunter and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 7–17, 1997. <https://doi.org/10.1145/266420.266426>.
- [55] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999. <https://web.archive.org/web/20211002203331/http://lpdwww.epfl.ch/fgaertner/pubs/TUD-BS-1999-02.abstract.html>.
- [56] M. Sirbu and J. D. Tygar. NetBill: an Internet commerce system optimized for network-delivered services. *IEEE Personal Communications*, 2(4):34–39, 1995. <https://doi.org/10.1109/98.403456>.
- [57] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pp. 8–26, 1996. <https://doi.org/10.1145/248052.248054>.
- [58] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966. <https://doi.org/10.1145/365230.365252>.
- [59] M. S. Miller, K.-P. Yee and J. Shapiro. Capability myths demolished. Agoric papers archive, 2003. <https://papers.agoric.com/assets/pdf/papers/capability-myths-demolished.pdf>.
- [60] P. Helland. Idempotence is not a medical condition. *ACM Queue*, 10(4):30–46, 2012. <https://doi.org/10.1145/2181796.2187821>.

- [61] M. R. Darby and E. Karni. Free competition and the optimal amount of fraud. *The Journal of Law and Economics*, 16(1):67–88, 1973. <https://doi.org/10.1086/466756>.
- [62] U. Dulleck and R. Kerschbamer. On doctors, mechanics, and computer specialists: the economics of credence goods. *Journal of Economic Literature*, 44(1):5–42, 2006. <https://doi.org/10.1257/002205106776162717>.
- [63] J.-C. Rochet and J. Tirole. Platform competition in two-sided markets. *Journal of the European Economic Association*, 1(4):990–1029, 2003. <https://doi.org/10.1162/154247603322493212>.
- [64] A. Hagiu and J. Wright. Marketplace or reseller? *Management Science*, 61(1):184–203, 2015. <https://doi.org/10.1287/mnsc.2014.2042>.