Agent Payments Protocol

An Open Protocol for Agent Commerce

OKX Payment team — Agent Payments Protocol Working Group
okx.com

v1.0 — April 2026

Agent Payments Protocol teaches agents to do business — a commercial relationship, not just an HTTP response.

— OKX Payment team, Agent Payments Protocol Working Group

Abstract

Agent Payments Protocol is an open protocol that lets a single AI run an **entire commercial relationship** — discover counterparties, negotiate scope and price, escrow funds, meter consumption, settle on-chain, handle disputes, split revenue, close the billing period — autonomously, with humans intervening only on exceptions rather than at every step. To make this possible, **Agent Payments Protocol expands the unit of interaction from a single transfer to a full commercial relationship**: four intents (`charge`, `escrow`, `session`, `upto`) cover the lifecycle of a deal, and a protocol-defined **Broker** — the off-chain orchestration service that mints payment objects, verifies credentials, and broadcasts settlement — absorbs the orchestration complexity that would otherwise land on every agent. Because the payment payload is **decoupled from any specific transport** (earlier agent-payment work embedded payment semantics inside a single HTTP response and was thus bound to HTTP), the same signed request and authorization flow unchanged over HTTP, XMTP, Telegram, Discord, Slack, Email, or a printed QR code, making **Agent-to-Agent (A2A) payment over an IM network** a first-class deployment shape. **Agent Payments Protocol** shares its EVM challenge/credential wire format with the Machine Payments Protocol (MPP).

Scope of this paper. This document presents the vision and design model of **Agent Payments Protocol**: the four-intent commercial envelope, the Broker role, and the A2MCP / A2A deployment shapes. Wire-format schemas, security analysis, fee model, and reference-implementation details are out of scope here and are maintained in the **Agent Payments Protocol** specification. This paper is intended to remain stable; subsequent revisions will be made in the specification rather than in this document.

Contents

1 Vision **3**

2 The case for Agent Payments Protocol **3**
2.1 A motivating scenario . . . 3
2.2 Why a payment rail is not enough . . . 4
2.3 Why now . . . 4

3 Where Agent Payments Protocol fits in the stack **5**
3.1 Where each protocol sits . . . 6
3.2 Side-by-side comparison . . . 6
3.3 Broker and x402 Facilitator: same slot, different scope . . . 7

4 Design Philosophy **8**
4.1 From payment to commerce . . . 8
4.2 Four design principles . . . 8

5 How Agent Payments Protocol Works **9**
5.1 Roles . . . 9
5.2 Deployment shapes: A2MCP and A2A . . . 9
5.3 Architecture in three pieces . . . 11
5.4 Reusing audited primitives . . . 11

6 The Four Intents **12**
6.1 `charge` — peer-to-peer instant transfer . . . 12
6.2 `escrow` — task custody with a dispute window . . . 12
6.3 `session` — streaming payment channel . . . 12
6.4 `upto` — pre-authorised metered deduction . . . 13
6.5 Intent comparison matrix . . . 13

7 IM-Native Delivery **13**

8 Built-in commerce features **14**
8.1 Splits — fee routing built into settlement . . . 14
8.2 Pluggable dispute resolution . . . 14
8.3 How `upto` prevents over-billing . . . 14
8.4 Separating signing from custody . . . 14

1 Vision

Recent agent-payment work has focused on *a human teaching an AI how to complete one transaction*. **Agent Payments Protocol** starts from the opposite premise: the AI is the merchant, the buyer, and the platform operator alike, with humans setting policy and budget rather than driving each step.

Concretely, **Agent Payments Protocol** can be summarised in a single equation:

Agent Payments Protocol = MPP EVM + four intent envelopes + Broker orchestration + cross-IM/HTTP transport.

The four terms map to the four parts of this paper: the MPP EVM wire format **Agent Payments Protocol** adopts is discussed in §3; the four intent envelopes are specified in §6; the Broker role is described in §5; and the cross-channel transport model is covered in §5.2 and §7. The remainder of §1 expands the design commitments behind that equation.

1. Agent autonomy is the goal. Earlier work made an AI capable of signing one transfer. **Agent Payments Protocol** makes an AI capable of running an entire commercial relationship: discover a counterparty, negotiate scope and price, hold funds in escrow, meter consumption, settle, dispute, split revenue, and close out — with humans intervening by exception rather than at every step.

2. The unit of agent commerce is a relationship. Earlier agent payments were modelled as a single transfer: one signature, one amount, one moment in time. Real commerce has more shape than that — negotiations carry scope and price, tasks carry acceptance and dispute windows, consumption streams carry a beginning, middle, and end. **Agent Payments Protocol** treats each commercial relationship as a first-class object with its own state machine, so escrow, metering, and dispute resolution are protocol features rather than workarounds.

3. Agent Payments Protocol is an open protocol. Every role here — Broker, Buyer Agent, Seller Agent — is defined by its interface, not by who runs it. There is no required vendor, no proprietary SDK, and no dependency on any single operator: any team can implement a compliant Broker, integrate **Agent Payments Protocol** into their own runtime, or build agent products that speak the protocol on their own terms. OKX's OnchainOS — its on-chain operating environment for agent skills, wallets, and identities — is one such runtime, with **Agent Payments Protocol** integrated out of the box.

2 The case for Agent Payments Protocol

2.1 A motivating scenario

Consider a single buyer-agent task: "plan and book my weekend trip." To finish the job, the agent must pay several counterparties, each in a different commercial shape:

- a **translation API**, priced per token, used to translate foreign-language hotel listings and local guides (*streaming, metered*);
- a **freelance trip planner** commissioned in a Telegram group for a custom itinerary, paid only on delivery (*escrow with dispute window*);
- a **hotel deposit** for a fixed price (*one-shot charge*);
- an **LLM call** summarising local restaurant reviews "of at most 5,000 tokens" (*capped metered deduction*).

Today, those four payments are four integrations on four different rails — a credit-card form, an HTTP `402` endpoint, an off-protocol Telegram bot, and a custom LLM-billing SDK — and three of them require the agent to drop out of its current conversational channel and call into HTTP. **Agent Payments Protocol** collapses all four into one protocol with one wire format and one settlement path: four *intents* on the same `challenge`/`credential` envelope, one for each shape above — `session` for the translation API, `escrow` for the freelance trip planner, `charge` for the hotel deposit, and `upto` for the restaurant-review LLM call — deliverable over whatever transport the agents are already using. §6 defines each intent in full.

2.2 Why a payment rail is not enough

The first wave of agent-payment work focused on the *moment of payment*: HTTP-embedded payment standards placed transfer semantics into an API; card networks exposed checkout flows to agents; major wallet vendors shipped agentic signing paths. Each solved one bottleneck: an agent can complete a single token transfer without human intervention.

Real commerce is not one `transferFrom`. It is:

- **Negotiation** — scope, price, deadline.
- **Escrow of funds** — locking when outcomes are uncertain.
- **Metering** — per token, per second, per call.
- **Release or refund** — based on delivery, dispute, or timeout.
- **Splits** — platform fee, creator royalty, agent operator cut, referral bounty.
- **Records** — an auditable on-chain trail of who paid whom for what.

When an agent negotiates a translation task, streams an LLM bill at the token level, or escrows a design job that a buyer must sign off on, one-shot `402` is too narrow in scope — it covers exactly one HTTP request-response, with no escrow, no metering, and no acceptance window — and a single `transfer` is too raw, since it carries no commercial semantics beyond "move tokens."

Agent Payments Protocol's thesis: agent commerce needs a protocol whose unit is *a commercial relationship*, not *an HTTP response*. That protocol must natively support escrow, metering, partial refunds, and splits — and its transport must not be bolted to HTTP.

2.3 Why now

Three preconditions enable **Agent Payments Protocol**:

Precondition	Status
Agentic wallets (autonomous per-action signing)	EIP-7702 deployed across multiple EVM chains; OKX agentic wallet live on X Layer
Sub-cent stablecoin settlement	Major stablecoins on L2 routinely transfer for sub-cent fees
Agent-native transports	A messaging stack has emerged across distinct slots: XMTP for IM-style agent messaging, A2A for inter-agent invocation, and MCP for agent-to-tool calls. **Agent Payments Protocol** treats each as a carrier rather than a substitute for the others.

Agent Payments Protocol is built on all three.

3 Where Agent Payments Protocol fits in the stack

Agent Payments Protocol builds on prior work in agent and on-chain payments. MPP is the wire-format substrate **Agent Payments Protocol** is built on; x402 is a peer protocol **Agent Payments Protocol** is structurally aligned with at the HTTP request layer, where their scopes overlap. This section maps where each protocol sits and how **Agent Payments Protocol** composes with them, then compares **Agent Payments Protocol** dimension-by-dimension against the most relevant ones.

At the wire level, every payment is a pair of structured messages. A `challenge` is the payment request issued by the Seller (through the Broker), specifying amount, token, recipient, and intent. A `credential` is the Buyer's signed authorisation responding to that challenge, which the Broker submits for on-chain settlement. These two messages are the unit of interoperability with MPP, and the carrier through which the four intent shapes are expressed.

3.1 Where each protocol sits

Protocol	Layer	How Agent Payments Protocol composes with it
x402	HTTP `402` middleware	x402 pioneered HTTP-native agent payments and is the right tool inside its scope. **Agent Payments Protocol** is structurally aligned at the request layer (`charge` can emit an x402-shaped `402`; on-chain settlement is **Agent Payments Protocol**'s own path) and carries the same idea into IM, QR, and offline channels where HTTP is not the natural carrier.
MPP	EVM payment method + session specification	**Agent Payments Protocol** directly consumes the MPP challenge/credential envelope and aligns with MPP EVM Session. Integrating MPP EVM is integrating **Agent Payments Protocol** `charge`.
ERC-8004	Agent identity registry	Seller agent addresses may reference an ERC-8004 registration. **Agent Payments Protocol** does not require it, but renders it on delivery cards when present, so identity work done in ERC-8004 surfaces naturally inside **Agent Payments Protocol** flows.
A2A / MCP	Agent-to-agent and Agent-to-tool transports	Both are first-class **Agent Payments Protocol** Seller surfaces. In the **A2MCP** shape the Buyer Agent consumes a priced HTTP service (often invoked through an MCP tool) and the `challenge` is returned over HTTP — typically in a `402` response or as an out-of-band URL. In the **A2A** shape the `challenge` rides in an A2A message as a `url` / `card` / `raw` delivery. Embedding **Agent Payments Protocol** directly inside MCP `tools/call` is a natural area for further design work.
Card-network agent checkout	card rails	Different design centre. Card rails serve fiat-anchored, dispute-via-bank flows; **Agent Payments Protocol** targets stablecoin-native, finality-instant settlement with on-protocol escrow. The two can coexist on the same merchant.

Positioning. Agent Payments Protocol is not a competitor to either x402 or MPP. It is the simultaneous result of three things: (i) adopting MPP's EVM grammar, (ii) layering four intent shapes on top, and (iii) productising the orchestration through a Broker whose deliveries can travel any IM. Where x402 already fits, **Agent Payments Protocol** defers to it; where the conversation moves off HTTP or past one round-trip, **Agent Payments Protocol** extends the same vocabulary.

3.2 Side-by-side comparison

The table below is a positioning map, not a scorecard. Each row is a design dimension; each column is the choice a given protocol made for its own scope. x402 and MPP are deliberately

narrower than **Agent Payments Protocol** in the precise sense that they target a smaller portion of the commercial surface: x402 is scoped to a single HTTP `402` round-trip, and MPP is scoped to the EVM payment-method wire format (challenge / credential / receipt, plus a session channel). Neither attempts to cover escrow, dispute resolution, or non-HTTP transport — those are simply not in their scope — and within the scope each one chose, each is the right answer. **Agent Payments Protocol** adds those additional dimensions on top of the same foundation rather than replacing it.

	Credit card	**x402**	**MPP**	**Agent Payments Protocol**
Finality	days	chain-dependent	chain-dependent	chain-dependent[1]
Revocability	chargeback within 120 days	not in scope — by design	not in scope — by design	via `escrow` dispute window
Transport	banks / Visa rails	HTTP-native	transport-agnostic spec	HTTP + IM + QR + offline
Intent shapes	auth-capture	one-shot	Charge + Session	Charge + Escrow + Session + Upto
Splits	handled outside the protocol	handled outside the protocol	per-method (`charge` only)	first-class, bps-native
Dispute	chargeback	out of scope	out of scope	optional arbitration
Agent-native	emerging via card-network agent extensions	yes	yes	yes, and IM-native

Table 1: Where each protocol fits. Credit cards are the comparator from the pre-agent world; x402 and MPP are the agent-payment foundations **Agent Payments Protocol** composes with. **Agent Payments Protocol**'s wider footprint reflects a wider scope, not a judgement of the others.

3.3 Broker and x402 Facilitator: same slot, different scope

Agent Payments Protocol's Broker and x402's Facilitator occupy the same architectural slot — the off-chain coordinator between Buyer and Seller (each an agent, or a human represented by one) — at different scopes. The Facilitator is stateless, right-sized for a single HTTP `402` round-trip; the Broker is stateful, because **Agent Payments Protocol**'s unit of interaction is a commercial relationship that can span multiple steps and several days.

Dimension	**x402 Facilitator**	**Agent Payments Protocol Broker**
Designed for	A single HTTP `402` round-trip	A commercial relationship that may span multiple steps
State	Stateless by design — correct for a single request	Persists challenges, `paymentId`s, vouchers, state machines
Lifecycle	Completes within one request	Spans `create` → `settle`/`close`, possibly days or months
Scope	Verify + settle	Verify + settle + delivery generation + arbitration + usage-report validation + channel maintenance
Transport	HTTP-native	Transport-agnostic: IM / HTTP / QR / offline

The role is named "Broker" because it is a commercial intermediary — one that may charge fees, hold funds across days, and potentially be licensed — which is a different shape from the

[1] Finality and on-chain fees depend on the settlement chain. The OKX reference Broker uses X Layer (∼200 ms finality, sub-cent fees), but **Agent Payments Protocol** is not bound to any specific chain. Broker fees are separate and out of protocol.

Facilitator's verify-and-broadcast scope.

4 Design Philosophy

4.1 From payment to commerce

Dimension	The payment era	The commerce era (Agent Payments Protocol)
Unit of interaction	One HTTP `402` response	One commercial relationship
Transport	HTTP	IM / HTTP / RPC / offline QR
Intent shape	Fixed price, instant	Charge / Escrow / Session / Upto
Trust model	Pay and leave; no recourse	Conditional release + dispute window
Splits	Handled outside the protocol	First-class `splits` primitive
State	Stateless per request	Persistent payment object with a state machine
Settlement timing	Synchronous with the request	Asynchronous: open → accumulate → report → close

4.2 Four design principles

1. The protocol is stateless; roles are stateful. **Agent Payments Protocol**'s wire messages (`challenge`, `credential`) carry no session memory. State is held by the *Broker* role — and *any entity* may implement that role, provided it honours the interface. Consequently the transport can be any "fire and forget" carrier: a payment URL in a Telegram DM and the same URL over XMTP are semantically equivalent, and the URL can even be printed on paper.

2. The signature is the source of truth. The `payload.authorization.from` field, recovered by ECDSA from the signature, is the design's source of truth for payer identity. Human-readable fields (name, description, avatar) are presentation only. A Broker is not expected to override signed fields with unsigned ones, and a Buyer is expected to verify the critical fields before signing.

3. Wire-compatible with MPP. The **Agent Payments Protocol** wire format is a superset of the MPP EVM challenge/credential, so any client that already supports MPP EVM natively supports **Agent Payments Protocol** `charge`. **Agent Payments Protocol** is not a competitor to MPP but a productisation layer on top of it, adding commerce primitives (deliveries, usage reports, arbitration) on top of the MPP EVM grammar.

4. Roles are substitutable; the protocol does not depend on any single operator. **Agent Payments Protocol** does not assume a unique Broker. Buyer and Seller each pick a Broker they trust; in multi-Broker scenarios the credential binds to a specific Broker public key. At the protocol level, any implementation that satisfies the interface is a legitimate participant.

5 How Agent Payments Protocol Works

5.1 Roles

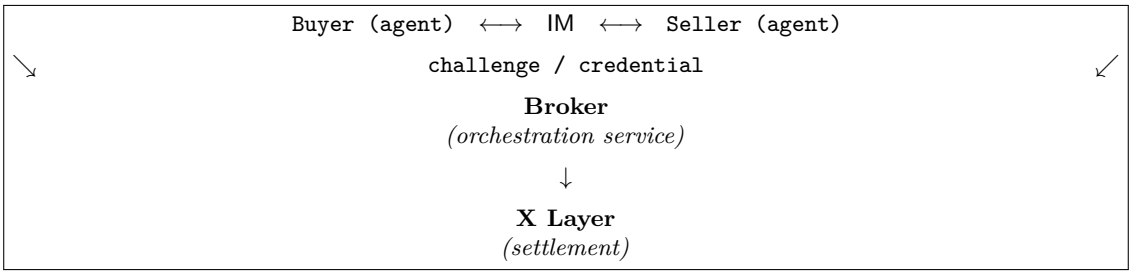

Figure 1: **Agent Payments Protocol** topology. The transport between buyer and seller is agnostic; the Broker is a protocol role, not a fixed vendor.

- **Buyer / Seller** — the real counterparties. A Buyer is always an Agent (or a human represented by one). A Seller may be another Agent reachable over an IM network (the **A2A** shape) or a priced HTTP service or tool the Buyer Agent consumes — often reached through an MCP tool on the Agent side (the **A2MCP** shape). **Agent Payments Protocol** treats both uniformly at the wire level — see §5.2.
- **Buyer Agent / Seller Agent** — the proxy programs (typically the payment-handling module in the agent's runtime) that parse challenges, sign, and poll state.
- **Transport** — any channel capable of carrying a URL, a card, a QR code, or a raw JSON blob. **Agent Payments Protocol** makes no demand on it: HTTP, XMTP, Telegram, Discord, Slack, Email, SMS, and deep links all qualify.
- **Broker** — *a protocol role, not a specific operator.* Any entity willing to perform the following duties is a valid Broker — a wallet vendor, an exchange, a DAO, a self-hosted service, or a counterparty itself:

  (i) accept and persist the Seller's payment request and mint a `paymentId`;

  (ii) produce the `challenge` structure and the distribution envelopes (URL / card / QR / raw);

  (iii) receive the Buyer's `credential`, verify the signature, match against the stored challenge, and recompute the nonce;

  (iv) broadcast on-chain, optionally sponsoring gas on the Buyer's behalf;

  (v) expose a status-query endpoint for both sides to poll.

  Buyer and Seller may each select a trusted Broker; when they differ, the `realm` field of the challenge coordinates settlement, and the credential can be bound to a specific Broker public key.
- **X Layer** — the reference settlement chain throughout this document. **Agent Payments Protocol** binds to no specific chain or token; any EVM chain that can settle stablecoin transfers authorised off-chain is a valid settlement layer.

5.2 Deployment shapes: A2MCP and A2A

Agent Payments Protocol is a single protocol with two common deployment shapes. They differ in *who plays the Seller role* and *which transport carries the* `challenge` — the four intents,

the `challenge`/`credential` envelope, the Broker role, and the on-chain settlement path are identical.

A2MCP — an Agent pays for a priced service or tool.
The Seller is an HTTP-addressable service, often discovered and invoked through an MCP tool on the Buyer Agent's side. When the Agent hits a priced endpoint, the service returns an **Agent Payments Protocol** challenge *over HTTP* — typically as an x402-style `402 Payment Required` carrying a challenge URL. The agent's payment module signs, the Broker settles, and the Agent retries the original call citing the `paymentId` (for `charge`) or draws from an already-open channel (for `session`). The *payment rail here is plain HTTP*. Embedding **Agent Payments Protocol** directly inside MCP — so a single `tools/call` could negotiate, settle, and return results in one round-trip — is a natural area for further design work.

A2A — an Agent pays another Agent.
The Seller is another Agent reachable over an IM network (XMTP, Telegram, Discord, Slack, Email, SMS). The Seller Agent asks its Broker to mint a payment, emits a `url` / `card` / `qrcode` delivery into the IM channel, and polls for settlement. Either or both sides may be user-facing; neither side needs an HTTPS endpoint, a webhook inbox, or a long-lived session.

The `challenge` carries no transport assumption. Whether the Agent fetches it from a `402` response on a priced endpoint or receives it as a URL pasted into an XMTP thread, the bytes are identical, the signature is identical, and the Broker settles it identically. A Seller that ships both a priced HTTP API and a Telegram bot therefore mints a single payment per transaction and selects which deliveries to emit on each channel.

Aspect	A2MCP	A2A
Seller runtime	Priced HTTP service (often invoked via an MCP tool on the Buyer side)	Agent on an IM network
Who initiates the call	Buyer Agent (by invoking the tool / hitting the endpoint)	Seller Agent (by pushing an invoice into the chat)
Challenge transport	HTTP (`402` response body or an out-of-band challenge URL)	IM (`url` / `card` / `qrcode`)
Typical intents	`charge`, `session`, `upto`	`charge`, `escrow` (with `splits`)
Typical latency	single round-trip, ∼seconds	minutes (`charge`) to days (`escrow`)
Analogue on today's web	Paywalled API endpoint	Invoice link in a DM

Table 2: A2MCP and A2A share the same **Agent Payments Protocol** wire format; they differ in who plays Seller and which transport carries the `challenge`.

Topology (A2MCP).

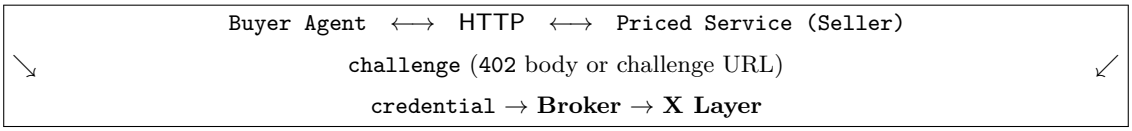

Figure 2: A2MCP. The `challenge` is returned over HTTP — usually as an x402-style `402` response carrying a challenge URL. The Buyer Agent may invoke the service through an MCP tool on its own side, but the payment itself rides on plain HTTP.

Topology (A2A).

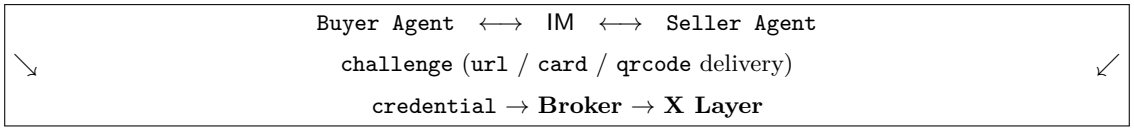

Figure 3: A2A. The `challenge` rides in an IM message — as a Broker-issued URL, a structured card, or a QR code. Neither side needs an HTTPS endpoint or a webhook inbox; both sides poll the Broker for status.

Agent Payments Protocol does not pick one shape over the other. A2MCP maps cleanly onto how Agents already consume priced tools on the web today; A2A extends the same protocol into peer commerce — negotiated tasks, escrowed deliveries, streaming peer-to-peer consumption, platform splits — over conversational channels where there is no HTTP endpoint to call. The rest of this document picks whichever shape best illustrates each intent.

5.3 Architecture in three pieces

At a high level, **Agent Payments Protocol** has three cooperating pieces:

1. **Off-chain Broker** — holds the order book and orchestrates the protocol lifecycle.
2. **On-chain settlement layer** — finalises every payment; held intents (`escrow`, `session`) route through an audited custody contract, while direct intents settle by submitting the Buyer's authorisation straight to the token.
3. **Configurable policy extensions** — for arbitration, fee splits, and payout routing.

The Broker absorbs the protocol's complexity so that Buyer and Seller can speak in plain commercial terms.

On-chain settlement, with custody where it's needed. Intents that do not need to hold funds — notably `charge` — settle by submitting the Buyer's signed authorisation directly to the token, in a single transaction with no intermediate contract. Intents that *do* need to hold funds — `escrow` (budget held until acceptance or dispute resolution) and `session` (deposit held for the duration of a streaming channel) — route through an audited on-chain custody contract that supports both modes under one design, so integrators deploy against one system rather than several. Both paths share the same wire envelope and the same Broker lifecycle, which is what lets **Agent Payments Protocol** present a uniform commercial surface across very different deployment shapes (A2MCP and A2A) without dragging integrators into the engineering complexity underneath.

Composable settlement workflow. Splits, royalties, and platform fees are not baked into the protocol — a marketplace, a DAO treasury, or an individual creator can each express their own settlement policy without forking anything. Dispute resolution is similarly pluggable: **Agent Payments Protocol** guarantees that the resolver's decision binds settlement, but *how* the resolver reaches that decision (human moderator, community vote, reputation system) is left open. The result is a protocol that does not pick winners in fee policy or arbitration — those are surface choices made by the participants, not by **Agent Payments Protocol**.

5.4 Reusing audited primitives

Agent Payments Protocol introduces no new signing scheme and no new custody design. Each intent reuses an existing, audited primitive whose shape already matches the commercial flow. Three of the four intents — `charge`, `session`, and `upto` — reduce to off-chain signatures over widely supported EVM signature standards already implemented by major wallets, so the

per-payment signing flow is one wallets already know: a one-shot transfer authorisation for `charge`, monotonic typed vouchers for `session` (so older vouchers are automatically obsolete and replay is impossible by construction), and a payment-bound pre-authorised cap for `upto` (so the cap cannot be reused elsewhere). Depending on the chosen primitive, first use may require a one-time on-chain setup (such as a token approval); per-payment signing itself is unchanged from what wallets already do today. `escrow` is the exception: holding a budget for the duration of a dispute window with conditional release cannot be expressed as a signature alone, so it is backed by an audited on-chain custody contract that the Buyer funds at order creation. We deliberately keep the specific standards and contract choices out of this paper so that **Agent Payments Protocol** remains free to track best practice as the underlying primitives evolve; the technical specification is the source of truth for the current implementation.

6 The Four Intents

Agent Payments Protocol defines four commercial intents. Each captures a distinct shape of commercial interaction.

The four intents below cover the commercial shapes that recur across current agent deployments: one-shot, escrowed, streaming, and capped. The set is kept small to simplify auditing of the wire format and the Broker state machine. It is not exhaustive: subsequent revisions of the specification may add further intents.

6.1 `charge` — peer-to-peer instant transfer

Use cases. Tips, gift cards, fixed-price goods, one-shot API calls with known price.

Shape. The Buyer authorises a single transfer to the Seller. One on-chain settlement, no holding period, no dispute window. This is the closest **Agent Payments Protocol** intent to a traditional one-click payment.

Mechanism. The Buyer signs a single off-chain transfer authorisation naming the Seller and the amount; the Broker submits it on-chain. There is no multi-step lifecycle to follow — one signature, one settlement.

6.2 `escrow` — task custody with a dispute window

Use cases. Freelance work (translation, design, code), any delivery requiring acceptance.

Shape. Funds are held by an escrow primitive between commit and release. The Seller submits delivery; the Buyer accepts or raises a dispute within a configurable window; an optional arbitrator can resolve disputes. Funds release on accept, refund on timeout or arbitration ruling.

Mechanism. The Buyer locks the budget on-chain when the order is created. The Seller submits delivery, which opens a configurable dispute window. Inside the window the Buyer can release (instant payout), dispute (an external arbitrator decides), or stay silent (the Seller self-releases once the window expires). Splits, if declared, are applied at the moment of release (see §8.1).

6.3 `session` — streaming payment channel

Use cases. LLM billing per token, API billing per call, continuously metered services, consumption-style subscriptions.

Shape. The Buyer opens a channel with a deposit; usage accrues off-chain through monotonically increasing vouchers; the channel closes with a single settlement that pays the Seller the

accumulated amount and refunds the residual to the Buyer. No on-chain activity during the stream itself, so metering can be arbitrarily fine-grained at near-zero cost.

Mechanism. The Buyer locks a deposit and then signs successive vouchers off-chain as usage accrues; each voucher records the cumulative amount, so only the latest one matters and stale vouchers can never be replayed. The Seller draws on the channel whenever it wants, typically once at close. Either side can initiate close, and any unused deposit refunds to the Buyer.

6.4 `upto` — pre-authorised metered deduction

Use cases. Single tasks with unknown amount but known upper bound — e.g. "generate an article of at most 5,000 tokens."

Shape. The Buyer pre-authorises a maximum amount. The Seller delivers, then reports actual usage; the Broker deducts the reported amount, bounded by the cap. The Buyer signs the cap and the Seller signs the usage report, so neither side can unilaterally inflate the bill.

Mechanism. The Buyer pre-signs a cap that is locked to this specific payment; the Seller signs the actual usage at delivery. Settlement deducts the minimum of the two — neither side can unilaterally inflate the bill, and the signed cap cannot be reused on any other payment.

6.5 Intent comparison matrix

	`charge`	`escrow`	`session`	`upto`
Amount known at sign time	yes	yes	no (unit price known)	no (cap known)
Settlement timing	instant	on accept / dispute resolution	on channel close	after Seller usage report
Dispute built in	—	yes	—	—
Typical latency	seconds	days	continuous	single request

Table 3: The four intents cover the commercial surface. One-shot, escrowed, streaming, and capped — each is a distinct composition.

7 IM-Native Delivery

The same `challenge` is wrapped in multiple envelopes. The Broker returns a `deliveries` array along with the payment; the Seller picks whichever subset fits the IM's capabilities.

Delivery	Format	Audience	Typical transport
`url`	Broker-issued payment link	general	any text channel
`card`	structured JSON, rendered as rich text	rich-text IMs	XMTP, Telegram, Lark
`qrcode`	PNG (base64)	offline, print, in-store	posters, storefronts, screens
`raw`	full challenge JSON string	technical integrators	agent-to-agent with shared SDK

Implication. A Seller invoicing a buyer in a Telegram group does not need to run an HTTPS server, host a `402` endpoint, or maintain a webhook inbox. The Broker generates a URL, the Seller pastes it into the chat, the Buyer's skill parses and signs, and the Broker closes the loop.

8 Built-in commerce features

A simple transfer settles one amount between two addresses. Real commerce involves more: a platform takes a cut, a referrer earns a bounty, a dispute may need a neutral third party, and most of the time the agent that signs is not the wallet that holds the money. **Agent Payments Protocol** promotes these into first-class primitives so that builders don't reinvent them per integration.

8.1 Splits — fee routing built into settlement

When a payment settles, **Agent Payments Protocol** can route it across multiple recipients in pre-declared proportions — a platform fee, a referral cut, a creator royalty — without post-hoc accounting. The Seller declares the splits up front; settlement moves each share to the right recipient in the same step that pays the primary Seller, removing the need for off-chain reconciliation or a separate batch-payout step.

8.2 Pluggable dispute resolution

For `escrow` payments, **Agent Payments Protocol** does not pick a single dispute resolution model. The protocol accommodates an external resolver — a human moderator, a community vote, a reputation system, or any other arbitration mechanism — so the Seller and Buyer can match the resolver to the kind of work being done. The protocol is designed so the resolver's decision binds settlement; *how* the resolver reaches that decision is left open.

8.3 How `upto` prevents over-billing

`upto` payments are governed by two signatures: the Buyer's, which caps the maximum the payment can ever deduct, and the Seller's, which reports the actual usage. Both are required for settlement. Neither side can unilaterally raise the bill, and the Broker enforces the cap during settlement: the Buyer cannot be charged above the cap, and the Seller cannot collect more than they have signed for.

8.4 Separating signing from custody

In a `session`, the wallet that holds the funds and the key that authorises each unit of usage do not have to be the same. The Buyer can delegate per-turn signing to a hot key or an agent-side key, while the main wallet retains the rights to deposit and to close the channel. This matches how real agents are architected — the agent holds a hot key for high-frequency action, but a cold wallet holds the money. **Agent Payments Protocol** encodes that separation into the protocol so the funded wallet never has to be on the per-turn signing path.

---

*Authored by OKX Payment team, Agent Payments Protocol Working Group. Released for feedback. Wire-format schemas, security analysis, and reference-implementation details are maintained in the **Agent Payments Protocol** specification. This paper does not constitute a security audit, and it does not constitute an offer of services.*