Paying Without Keys:
Request-Scoped Settlement for Autonomous Onchain Agents

fold research

research note rn-02 · Base mainnet · settlement over x402

Abstract

We describe how an autonomous onchain agent settles for the data and compute it consumes without holding an API key, without a subscription, and without a human authorising each purchase. In place of a durable credential, the agent presents a fresh, single-use, cryptographically bounded payment authorisation at the moment a service demands one, and the network verifies it before any work is performed. We motivate why identity is the wrong primitive for autonomous software, specify a two-round-trip payment exchange in which the agent signs a spending *ceiling* rather than an exact amount, and report the frictions that surfaced deploying the scheme on a live mainnet—none of which live in the cryptography, and all of which live in the layers around it.

1 Introduction

An agent that reasons about markets is only as useful as the information it can reach. It needs prices, structure, funding, news, and the output of models larger than itself, and it needs them at the instant a decision is being formed, not on the schedule of a billing cycle. The conventional way to grant that reach is an API key: a long-lived secret, provisioned in advance by a human, that identifies an account already in a billing relationship with each provider. This arrangement is invisible to most software because most software is operated by the human who holds the key. An autonomous agent has no such human standing beside it at the moment of need, and a secret it must carry is a secret it can leak.

The mismatch is structural rather than incidental. A key asserts *who you are* and defers the question of *what you may spend* to an account balance settled later. An agent needs the opposite: it should assert nothing about identity and prove, in the same breath as the request, that a specific, bounded payment is authorised for exactly this call. The unit of trust must shrink from the account to the individual request. This note describes a payment scheme that makes that shrink concrete, and reports what building it on a live mainnet surfaced.

2 The Problem with Keys

A credential model assumes a durable party that can be billed. It works because a human onboarded once, agreed to terms, attached a card, and now carries a secret that stands in for all of that history. Every property that makes this convenient for a person makes it hazardous for an agent. The secret is long-lived, so its exposure is catastrophic and permanent until rotated. It is coarse, granting access to everything the account may do rather than to the one call at hand. And it is provisioned ahead of time, which means an agent cannot reach a service its author never anticipated without a human first returning to the loop to create an account and paste a key.

Property	API key	Request-scoped
Lifetime	long-lived	single call
Scope	whole account	one payment
Provisioning	human, ahead	on demand
Bound	opaque, server	signed, legible
Leak radius	total	one amount

Table 1: The primitive an autonomous agent requires, against the one it inherits from human software.

The deeper cost is that a key relocates the spending limit to a place the agent cannot see. Authority lives in a dashboard, a plan tier, a rate limit—server-side state the agent neither controls nor can reason about. When the agent asks whether it can afford a call, the honest answer is that it does not know; it can only try and observe a rejection. For a program whose entire discipline rests on knowing the bound of its own authority before it acts, this is the wrong shape.

2.1 What an Agent Needs Instead

The requirement inverts cleanly. Rather than a durable identity billed later, the agent should present a fresh, single-use, bounded authorisation at the moment of request, prove it cryptographically, and carry no standing secret between calls. The bound should be legible to the agent before it signs, so that affordability is a fact it establishes rather than a rejection it discovers. And the whole exchange should complete inside one request cycle, so that reaching a new service requires no human and no prior account. Table 1 states the contrast the rest of this note builds on.

3 The Exchange

The scheme begins where a normal request ends. The agent calls a resource as if it were free. The server, finding no payment attached, answers not with a denial but with an invoice: an HTTP `402` carrying, in a header, the exact terms under which it will serve the call—the asset it accepts, the network it settles on, the address it pays to, and a ceiling on the amount. Nothing about the agent's identity is requested, because none is relevant. What the server has published is a price and a boundary, and the agent now has everything it needs to decide.

If the terms are affordable and within mandate, the agent signs. The signature is not a login; it is a typed

Stage	Actor	Guarantee
402 invoice	server	terms before spend
Authorise	agent	a ceiling, no more
Verify	facilitator	signature, bound
Settle	contract	$\leq a_{\max}$, or revert

Table 2: The four stages of a keyless payment, and the invariant each one holds.

authorisation that says, in effect, *this account permits a transfer of up to this amount to this recipient for this request, and no more*. The agent attaches the signed authorisation to a second, identical request. This time the server, before running any work, hands the authorisation to a facilitator that verifies the signature and settles the transfer on chain, then returns the result the agent asked for in the first place. Two round trips, no key, no session, no human.

3.1 Why the Ceiling Matters

Not every price is known before the work is done. A fixed-cost lookup can be quoted exactly, but a call whose cost depends on how much it produces—the tokens a language model emits, the depth of a search—cannot. A scheme that demanded an exact amount up front would force the agent to over-commit to a guess or the server to refuse metered work entirely. The scheme instead lets the agent authorise a *maximum*, and lets the server settle the true amount, which may be any value at or below that ceiling:

$$s_{\text{settled}} \;\leq\; a_{\max}, \tag{1}$$

where $a_{\max}$ is the amount the agent signed. The agent's exposure is capped at what it signed; the server is paid for what it actually spent. A fixed-price data call sets $a_{\max}$ equal to its price and behaves exactly like an exact payment; a metered model call sets a generous $a_{\max}$ and settles far beneath it. The agent reasons about only one number—the worst case it has agreed to—and never has to predict the true cost of work it has not yet seen.

3.2 Where the Boundary Is Enforced

The authorisation is meaningless unless something outside the agent enforces it. That something is the token contract and a settlement proxy, not the server and not the model. The signature authorises a transfer under a standard permit; the on-chain contract is what actually moves funds, and it will move no more than the signed permit allows and send them nowhere but the named recipient. The server cannot inflate the charge, because it does not hold the authority to; it can only present the settled amount to a facilitator, which the contract will reject if it exceeds the ceiling or redirects the funds. The mind proposes the payment; the chain disposes of it within the exact bound that was signed.

The agent signs a message, not a transaction it must fund with gas on every call. Settlement is submitted by the facilitator, which means an agent can pay for a service while holding only the token it is spending—no separate balance to manage for the act of paying itself.

Friction	Lives in	Resolution
Unreadable terms	browser/CORS	expose headers
Throttled reads	rpc endpoint	dedicated node
Scheme drift	facilitator	match schemes

Table 3: Three frictions a live surface surfaces, none in the cryptography, all in the layers around it.

4 From Scheme to Mainnet

A scheme is a claim; a mainnet is a verdict. Serving the scheme over a real network, to real wallets, for real value, surfaced a set of frictions that no specification anticipates—most of them not in the cryptography but in the layers around it. They are worth recording precisely because they are the parts a reader building the same thing will meet, and the parts a specification leaves out.

4.1 The Bound Must Be Legible to the Caller

A browser will not let a page read a response header unless the server explicitly permits it. The invoice that carries the payment terms is a header; a client that cannot read it sees a bare rejection and cannot form a payment at all. The scheme is silent on this because it lives one layer below the browser's rules, but a deployment that ignores those rules fails before any signature is attempted. The correct posture is to make the payment headers legible to any origin and to reflect whatever headers a client declares it will send, so that a caller in a browser is never blind to the very terms it must satisfy.

4.2 Reading the Chain Is Not Free

Before an agent can sign, it must know its own balance and whether its account has already granted the settlement contract permission to move funds. Those are reads against the chain, and a default public endpoint answers them until it does not—under any real traffic it rate-limits, and the payment stalls not because anything is wrong with the payment but because the reads that precede it were throttled. The lesson generalises: the scheme's correctness lives in the signature, but its *reliability* lives in the infrastructure that feeds the signature, and that infrastructure must be provisioned as deliberately as the keys it replaces.

4.3 Verification Is a Distinct Authority

A server can publish an invoice in a scheme its own verifier does not yet settle. The two are separate systems—one advertises terms, the other checks and settles them—and they can drift. When they do, an agent signs a perfectly valid authorisation and receives, in place of its result, a second invoice: the verifier declined the payload it was handed. The failure is honest but confusing, because every visible step succeeded. It records a real requirement: the scheme a surface *advertises* and the scheme its facilitator *accepts* must be the same scheme, and confirming that is a deployment step, not an assumption.

5 Conclusion: An Account per Call

The result is a small inversion with a large consequence. The unit of trust is no longer an account that persists

between requests but an authorisation that exists only for one. An agent reaches a service it has never used before with no onboarding, because there is nothing to onboard; it proves it can pay in the same message that asks; and it carries no secret away, because there is nothing to carry. A leaked message is worth exactly the one bounded amount it authorised and nothing after. The reach of an agent stops being a function of what a human provisioned in advance and becomes a function of what it can afford, decided call by call, against a bound it signed itself.

This is the payment layer the surrounding system was built to assume: a mind that can act on the world without a human holding its purse. The scheme does not make the agent trustworthy—nothing does—but it ensures that whatever the agent decides, it may only spend what it was permitted to spend, one request at a time, with the boundary held where the agent cannot reach it.