Cleargate Finance

Anton Holovchenko (golvcenkoanton@gmail.com)

LinkedIn · X · Telegram

October 2026

The stablecoin-native authorization layer for autonomous agent payments. Whitepaper v1.2, public edition.

1. Introduction

Autonomous agents can already pay for things, but nobody stands behind those payments. Protocols such as x402, ACP and AP2, settled in stablecoins, have made machine-to-machine payment simple. What they have not provided is a party that takes responsibility when a payment goes wrong.

Today an agent's purchase is a push payment: irreversible, unverified and unbacked. If the agent is hijacked, the money is gone. If the merchant never delivers, the money is gone. When a purchase is disputed, no one in the current stack accepts the loss, so the merchant absorbs it, and without the fraud signals that used to protect them.

Cleargate Finance sits on the buyer's side of the transaction. An owner connects an agent to an account, funds it and sets spending rules, without ever touching a wallet, a chain or a gas token. Every payment is authorized before it executes. We check it against a signed mandate, a counterparty risk model and the task the agent was actually given, and we record every decision in plain language, including the payments we blocked.

This works on day one with no merchants integrated. The checks run before the money leaves the account, so they protect an agent paying any endpoint on the open web. That is why we do not need a two-sided network before the product is useful to someone.

Merchant-side features come second: instant netted settlement with no gas, and a *hold* that keeps a payment back until delivery is confirmed and refunds it automatically if delivery never comes. A hold only controls the timing of money we already hold, so it puts none of our capital at risk.

Guaranteeing a disputed payment out of our own capital comes later still, and it is not part of the launch. A guarantee cannot be priced without real loss data. From the first transaction we therefore measure what a guarantee would have cost, and we sell nothing against that number until it is real.

2. The problem

The mechanics of agent payments are solved. What is missing is someone who is responsible for the outcome.

What is already solved

Transport protocols give agents a standard way to pay at no protocol fee. Stablecoins settle in seconds for a negligible cost. A dozen vendors sell wallet and key management, and identity registries let an agent carry a name. None of this is a place where a new company should compete.

Liability has no owner

In every proposed agentic commerce protocol the merchant stays the merchant of record. They carry the risk of an action no human took, and they lose the signals that used to protect them: device fingerprints, browsing behaviour, geolocation and card-present checks. Payment schemes will not take that liability, and neither will consumers, issuers or model providers. Regulation E and the EU AI Act cover only part of the ground and leave gaps around authorization and liability.

Stablecoin rails make this worse. Payments are final and cannot be reversed, so there is no chargeback to fall back on.

The current stack also treats two separate questions as one. Trusted Agent Protocol establishes whether an agent is real. Tokenized credentials establish what it may spend. Neither says what happens when a purchase is disputed.

A protocol cannot answer that, because a protocol has no balance sheet. Only a legal entity with capital can agree to absorb a loss. That is the position we intend to reach in time, and it is not the launch product. Underwriting needs loss data that does not exist yet, so we begin with the control layer that prevents losses and the measurement that will later let us price them.

Security is an open problem

Agents are steered by language, so indirect prompt injection is a permanent attack surface and not a bug waiting for a patch. In the Agent Security Bench, mixed attack strategies reached an average success rate of 84% across 13 model backbones. Separate research tested eight defences against indirect prompt injection and bypassed every one of them with adaptive attacks. A model cannot reliably tell trusted instructions from untrusted content when both arrive as text.

The exposure is already real. In May 2026 an attacker hid a transfer instruction in Morse code in a public reply. A model decoded it, a connected agent treated the output as a command, and about $175,000 moved to the attacker. Nothing in the chain had a transaction limit, a second confirmation or anomaly detection.

Our posture follows from that: assume the model will be compromised, and limit what a compromised model can do. We do not claim to prevent prompt injection. We contain it, so that a compromised agent cannot spend beyond its mandate.

Trust has not arrived

A June 2026 report found that only 14% of consumers trust AI to complete purchases without verification. Consumer agent commerce is not close, which is why we start with businesses that run agents.

Merchants are working blind

Merchants used to see impressions, clicks and funnel drop-off. Agent traffic produces almost none of that, so it is hard to justify investing in it even where payments work. Today a merchant can block agent traffic or accept it blind. We offer a middle setting: accept backed agents and refuse unbacked ones.

The market is small today

Agent payment volume is measured in the tens of millions of dollars a month, and a meaningful share of that is not genuine commerce. Qualified merchants number in the low thousands. This is not a large market yet, and our plan accounts for that. For the first eighteen months, revenue comes from subscriptions that do not depend on transaction volume. Guarantees are an option on 2028 to 2030 activity. Any plan that needs agent commerce to be large in 2026 fails.

3. What Cleargate Finance is

Cleargate Finance controls what an autonomous agent is allowed to spend, and gives both sides of a transaction a reason to trust a purchase that no human watched. In the near term it is the trust and authorization layer for agent payments. Over time the aim is a financial platform for the machine economy, moving from authorization to holding funds, then credit, then underwriting. Stablecoins and x402 are how we make the near term possible, and a customer never needs to know they are there.

Three outcomes define the system:

1. **An agent cannot spend beyond the authority its owner gave it.** This is enforced below the model, where a compromised prompt cannot reach it. It needs no merchant, no network and no consent from the counterparty, which is why it comes first.
2. **A buyer does not pay for something that never arrived.** Where we control settlement timing, payment is held until delivery is confirmed and returned if it is not. For a one-off payment to an endpoint on the open web, the protection is everything that runs before the money moves, and we say so plainly instead of implying recourse we do not have.
3. **Both sides can see and prove what happened.** Every decision, including every blocked payment, is recorded, signed and readable.

Absorbing disputed losses on our own balance sheet is a fourth outcome we intend to reach and do not claim today. Section 6 explains why.

What each party gets

Party	What they get
Agent owner	An account with agents in it. Each agent has spending rules, a share of one pooled balance and a readable log of what it bought and what it was stopped from buying. No wallet, no chain, no gas.

Party	What they get
Developer	One SDK that wraps the agent's HTTP client and handles any payment request automatically.
Enterprise	A policy and audit layer over agent spending, with attribution, approval workflows and exportable records.
Merchant	Instant settlement with no gas or chain handling, fiat payout if wanted, hold-based protection against non-delivery, and a place in a directory that buyers' mandates prefer.

This table is also the build order. Buyer-side control is useful with zero merchants integrated, so it comes first. Merchant features need a merchant to integrate, and the buyers routing payments are what make that worth a merchant's time.

What it is not

- **Not a payment rail.** It routes over x402 and, later, ACP, AP2 and card rails.
- **Not a wallet.** Key custody is a commodity, and we buy it from a specialist provider.
- **Not an identity registry.** We read external registries as weak signals.
- **Not a marketplace.** We build a directory that reports who payees are, what they charge and whether they deliver (Section 10). We do not take a cut of matching buyers and sellers, hold inventory or own the commercial relationship.
- **Not a treasury or trading product.** The authorization engine is generic and could one day serve agents acting in DeFi, but that would be a separate product and is not being built now.

Where it sits in the market

Wallets with spend controls are a crowded category, and most of it is built around card rails. We do not position Cleargate as an agent wallet with limits. Our lane is the combination of three things, and the combination is harder to copy than any one of them.

1. **Stablecoin-native, on x402 and Base.** This is less contested than cards and fits the segment that is actually transacting today: sub-dollar, high-frequency, machine-to-machine payments. Card-focused products optimise for a $40 consumer purchase. We optimise for eight thousand payments of half a cent.
2. **Buyer-side control and merchant-side trust together.** Most of the field picks one. Holding both turns a control panel into a network, because delivery outcomes seen on the merchant side are what make the buyer-side counterparty check worth anything.
3. **A path to bearing liability.** This takes capital and a loss book, which a protocol cannot have. It is a future differentiator and we describe it as one.

In one line: the stablecoin-native authorization layer that controls and protects agent payments.

4. How it works

An owner sets rules once, agents then pay freely inside those rules, and Cleargate checks each payment before it moves.

Core concepts

The platform separates what something is from what it does. An account is what a business is. Paying and receiving are things it does.

Concept	Meaning
Principal	The person or legal entity ultimately responsible. Liability and identity checks attach here.
Account	The container for a balance, users, agents and registered services. It is neither buyer nor seller by nature.
User	A human who logs into an account at a defined permission level.
Agent	The autonomous actor that initiates payments. It holds a mandate, not money.
Mandate	A signed grant of spending authority: limits per transaction, per day and per month, permitted categories, time to live, delegation depth and the threshold for human approval. It is versioned and verified before every payment.
Counterparty	Anyone an agent pays, whether an external endpoint or another account. Reputation attaches here.

"Merchant" is not an entity. It is an account acting as payee for one transaction. Any account can be payer or payee, so an agent paying a merchant and an agent paying another agent follow the same path through the system.

The agent owner's journey

Setup takes four steps:

1. Sign up with a company email. No crypto knowledge is needed.
2. Fund the account by card or bank transfer.
3. Register an agent and set its rules: maximum per purchase, monthly cap, permitted categories and approval threshold.
4. Copy an API key into the agent's configuration.

After that the agent works and the owner does nothing. The dashboard reads in plain language:

Time	Event	Status
2:14pm	Paid $0.40 to weather-api.com for forecast data	Delivered
2:19pm	Paid $2.00 to news-api.io for article search	Delivered
2:31pm	$85 to newsite.xyz: over the $20 limit, domain 3 days old	Blocked

The merchant's journey

A merchant registers, adds one call to its server that tells Cleargate when delivery succeeds, and gives payout details. Agents then pay. Settlement arrives daily, netted into one on-chain transfer, with no gas to fund and no chain to handle. Payout can be in stablecoin or, through an off-ramp, in fiat.

Where a payment is held pending delivery, confirming delivery releases it. A merchant that delivers is paid, and one that does not was never paid.

We do not sell netting to merchants as a feature, because it mostly lowers our own costs. We also do not promise that we absorb disputed losses. That is a later product, and it depends on real loss data (Section 6).

One account, three ways to use it

There is a single signup. An account becomes a buyer by connecting agents, which needs funding and mandates. It becomes a seller by registering services, which needs a payout destination and delivery confirmation. It can do either or both, on one account with one balance, and the role is decided per transaction.

Business machinery appears only when it is needed: team roles when a second user is invited, tax details when an invoice is required, identity verification when volume crosses a threshold. A buyer who later starts selling adds a service to the same account, with no migration.

Two settlement flows, one set of checks

Whether the payee has integrated with us changes how the money moves. It does not change what is checked before it moves, which is why the product works before any network exists.

	On-platform payee	Off-platform payee
Settlement	Netted: many payments, one on-chain transfer, using x402's batch-settlement scheme	One on-chain payment per purchase, using x402's exact scheme
Buyer-side checks	Mandate, counterparty, intent, behaviour	Identical

	On-platform payee	Off-platform payee
Protection	Pre-payment checks plus a hold until delivery is confirmed	Pre-payment checks only
Recourse after payment	Hold expiry refunds automatically	None. A raw x402 payment is final

Holds need control over timing, so they are available only on-platform. Once an exact payment lands on chain there is nothing left to withhold. For off-platform payees the console marks the payee as unverified and not covered. Where the exposure justifies it, we can proxy the request so that delivery is observed instead of assumed.

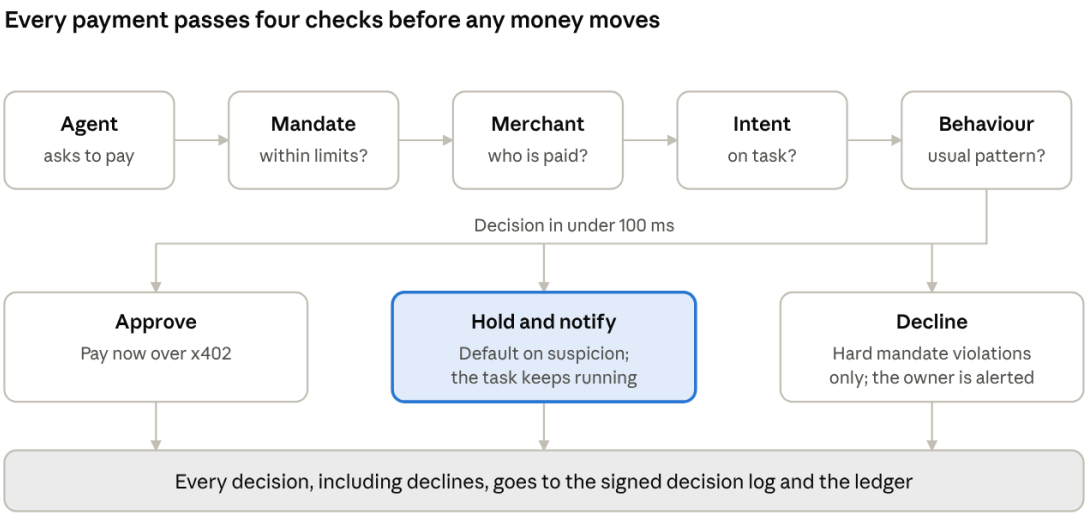
Off-platform payees are not a hole in the product. They are the top of the funnel, and Section 12 explains how their traffic converts them.

When things go wrong

Situation	What happens	Who bears the loss
On-platform merchant takes payment and delivers nothing	The payment was held, not released. The hold expires, the buyer is refunded and the merchant is flagged across the network	Nobody
Off-platform merchant takes payment and delivers nothing	The payment was final. The outcome is recorded against the counterparty and in our loss-observation ledger, and the console says which payees fall outside hold coverage	The buyer, as disclosed
Agent is hijacked and attempts a large transfer	Blocked at the mandate check, and the owner is alerted	Nobody
Owner disputes a purchase their agent made	We check the signed mandate and the decision record. Inside the mandate, the owner authorized it. Outside it, we approved in error, which is recorded as an observed loss and handled case by case	Set by the record
Coordinated refund farm against one merchant	The pattern is visible across all merchants at once and is blocked centrally	Nobody

Situation	What happens	Who bears the loss
Agent retries after a timeout	An idempotency key returns the original result, and no second payment is made	Nobody

A payment, step by step



payment flow · four checks, three outcomes

The four checks run in order. The result is an approval, a hold or a decline, and all three are logged.

5. The authorization engine

Four checks run in sequence before any money moves: mandate, merchant, intent and behaviour. The decision comes back in under 100 milliseconds.

Mandate check

The owner's spending rules are signed and evaluated on the server. Model output is treated as a request and never as an instruction.

This is the control that contains hijacking. An injected instruction may still persuade the model, and we make no claim to detect injection. But the resulting payment meets a wall: an amount above the cap, a category that is not permitted, an unknown destination or a size far outside the pattern. A catastrophic loss becomes a declined transaction and an alert. Because owners sign their mandates, even a breach of our own database cannot create spending authority.

Merchant check

Owners never list merchants in advance. An agent finds most of what it buys at runtime, so pre-approval would make the system unusable. Card networks solved this long ago: the network knows a shop the

cardholder has never heard of, because thousands of others have paid there. Identity and risk come from the network and not from the account holder.

Mandates therefore describe classes and limits, never specific payees. A rule might read: data and inference APIs, maximum $20 per call, hold anything unproven. No URLs appear in a mandate. Merchants are scored in four layers.

1. **Passive signals, available instantly.** Domain age, certificate age, DNS and hosting reputation, threat feeds, price against category norms, and whether the advertised payment details match the live endpoint. The most useful signal is provenance: did the URL come from the agent's original task, or did it appear mid-session in fetched content? That is the main tell for injection and SEO-poisoning attacks.
2. **Network history.** Once any participant pays a merchant, everyone after benefits. The thousandth customer gets a score built by the first 999.
3. **Merchant opt-in.** Merchants who want instant settlement register and, later, post a bond. A legitimate new merchant can leave cold start in days.
4. **The hold, where available.** An unknown on-platform merchant is held, not declined. Funds are committed, delivery is verified, and then the funds release. Exposure stays close to zero whoever the merchant turns out to be, the agent still finishes its task, and the payment produces the first data point for layer 2.

An off-platform payee has no hold, because an exact payment is final once it settles. Layers 1 to 3 still run, the decision is still recorded and the counterparty still builds a delivery record. The protection is entirely before payment, and the console says so.

Two asymmetries make this work. Trust accrues slowly while distrust spreads instantly: one confirmed non-delivery flags a merchant across the network within seconds. And the cost of being wrong is bounded by the hold, not by the size of the payment.

Intent check

We compare the stated purpose of a payment with the task the owner assigned. Hijacked agents typically make off-task payments, such as a research agent buying a licence or paying an address unrelated to its work. Off-task spending is a cheap, strong signal, and spotting it needs no analysis of the attacker.

Behaviour check

Agents are far more predictable than people. Human card spending is noisy, which is why bank fraud models need huge datasets and still flag too many good customers. An agent runs a loop with the same endpoints, similar amounts and a similar cadence. One that has paid five known endpoints ten thousand times and then sends forty times its usual amount to a domain registered days ago is easy to catch. A z-score and a novelty flag do it, with no machine learning.

Bank fraud features do not carry over, since there is no device fingerprint, geolocation or browsing history. Ours are machine-native: merchant age and history, price deviation, delivery rate, spend relative to task, delegation depth, time since the mandate was issued and the rate of new merchants per session.

Phase	Capability	Why
Launch	Deterministic rules: caps, categories, velocity, friction for new merchants	Delivers most of the value, and it is the only version an auditor can follow
Months 3 to 6	Per-agent statistical baselines	Strong signal, no labelled data needed
Months 6 to 12	Merchant scoring from delivery outcomes	Builds the proprietary dataset
Year 2 onward	Supervised models	Needs thousands of confirmed loss events

Operating constraints

Latency. Nothing slow runs in the request path: no chain calls, no model inference and no external HTTP. Merchant scores are computed offline and cached.

False positives cost more than they do with cards. A blocked human tries another card. A blocked agent fails mid-task with nobody watching. On suspicion, the default is to hold and notify, not to deny. Outright decline is reserved for hard mandate violations, and we treat the false-positive rate as a headline metric from day one.

What the system does not stop

We state these plainly, because overclaiming would be a problem for any future guarantee.

- Prompt injection itself. We guarantee containment, not prevention.
- A trusted merchant quietly letting quality slide while still technically delivering.
- Subjective quality. Verification covers conditions a machine can check, such as status, schema, hash and completion. "Was the analysis good?" is not one of them.
- Compromise of the agent runtime, including malicious tools and poisoned tool descriptions. That is an agent-security problem rather than a payments one.
- Poor decisions inside policy. We bound authority and do not improve judgement.

6. Disputes, protection and liability

Cleargate launches with a settlement mechanic that protects buyers at no capital risk, and with a measurement of what a guarantee would cost. It does not launch with a guarantee.

Most disputes should not exist

A card network learns about a problem weeks later, when a cardholder reads a statement. By then the merchant has shipped, the money has moved, and both sides argue from partial records. We stand inside the transaction as it happens. We hold the mandate that was in force, the stated purpose and the delivery confirmation.

Dispute	Why it should not occur	What is left
"The agent shouldn't have bought this"	The purchase was checked against a signed mandate before approval. Inside the mandate, the owner authorized it and the record proves it. Outside, we made the error	A lookup, not a debate
"The merchant didn't deliver"	On-platform, the payment was held and release needs confirmation, so expiry refunds it. Off-platform, the pre-payment checks are the protection and the payment is final	Near zero on-platform, and disclosed off-platform
"Delivered but useless"	This cannot be designed away, because it needs human judgement	The only category that needs a real process

The operating metric is observed loss as a share of routed volume. Card fraud runs at roughly 0.05% to 0.1%. Above about 0.5%, a guarantee business would not work. We track the number from the first transaction.

What ships at launch

Two things ship, and neither risks capital.

The hold. In the netted on-platform flow we control when money reaches the payee. A payment can be committed but not released, checked against delivery, and returned automatically if delivery never comes. That makes an unknown merchant safe to deal with. It costs nothing to offer, because the money being withheld is money we already hold. Controlling the timing of money you hold is not underwriting, and it involves no reserve, no coverage promise and no absorbed loss.

The loss-observation ledger. From the first transaction we record internally every event that would have been a covered loss under a guarantee: non-deliveries, disputes and approvals that proved wrong. Each record carries the payee, category, amount, reason and the check that passed it. Nothing is paid out against these records, and the console never shows a customer a number that implies coverage.

The measurement is nearly free, and the data cannot be recovered later. A competitor that starts two years from now can copy a rules engine in a quarter. It cannot copy two years of observed delivery failures.

Guarantees: later, and only on evidence

A guarantee cannot be priced without real loss data, so offering one at launch would mean underwriting blind. One badly priced quarter can end a company. Nothing in this part is a build commitment. It describes the shape of the product if the data supports it.

Losses would be absorbed in four layers, each there so the next is rarely touched.

1. **The hold.** The payment was never released, so there is no loss. Most cases should end here, and it is already live.
2. **Netting.** A refund owed is deducted from the payee's next payout. This works like a chargeback without needing reversibility.
3. **A merchant rolling reserve.** A trailing percentage of payouts is retained as the payee's own clawback buffer.
4. **Our capped reserve.** The rare remainder, capped per transaction and per month.

The coverage boundary is set before anything is sold. A guarantee would cover:

- Verifiable non-delivery under conditions a machine can check
- Our own approval errors, meaning a payment we approved that the mandate did not permit

It would not cover:

- Subjective quality, such as "the analysis was poor"
- Spending the owner authorized that turned out to be unwise
- Off-platform payments we could neither hold nor verify
- Anything above the stated per-transaction and aggregate caps

We would fund the first reserve ourselves, as a customer-acquisition cost and not as profit. Merchant-funded mechanisms come next. Once there is a real loss book, a reinsurer would write the policy and Cleargate would act as a managing general agent, pricing and underwriting the risk while the reinsurer holds the capital. Fees would be priced from observed risk and never set flat.

We would start with very low caps and widen them only on proven loss data. We would never guarantee an amount that a bad month could not survive, and we would never widen a limit because a customer asked.

Why a protocol cannot copy this

A protocol has no balance sheet and cannot accept liability. Only a capitalised legal entity can. That is why the loss dataset, which is what unlocks reinsurance, is the company's core long-term asset, even though it earns nothing on the day it is collected.

7. Architecture and platform security

The system keeps slow and unreliable things out of the payment decision, and it keeps an auditable record of everything that happens.

Design principles

- **Start as a monolith.** One service with clean internal modules, split only when a part clearly needs separate scaling. Chain interaction is the one early exception. It runs as a queue-driven worker, so unreliable blockchain connections never enter the decision path.
- **Nothing slow in the request path.** Scoring is computed ahead of time and cached. The hot path reads the cache and evaluates rules.

- **Rails are adapters.** The authorization engine never knows what a chain is. It approves or declines, and a separate layer decides how value moves.

The ledger

The ledger is double-entry. Every movement creates two rows that sum to zero. Rows are append-only, and corrections are new reversing entries, never edits or deletes, so the state of the system can be rebuilt for any past moment. An hourly job sums every entry and pages us at once if the total is not zero. Payments authorize against the ledger in under 100 milliseconds, whatever the block time.

Two safeguards come straight from payment-systems practice.

- **Idempotency.** Every money-moving request carries a client-supplied key that is stored with its result. If an agent retries after a timeout, it gets the original result back and no second payment is made. Without this, a retry loop can create thousands of duplicate payments in minutes with no attacker involved.
- **Safe concurrency.** Balance reads on the money path use locking or atomic conditional updates, so two simultaneous requests cannot both spend the same last $10.

Settlement

Netting is not a Cleargate invention. It is x402's official batch-settlement scheme. The buyer deposits once into an on-chain escrow and signs an off-chain *voucher* for each request, and every voucher carries a new running total that supersedes the last. The seller or facilitator redeems many vouchers in a single on-chain transaction. Our rail adapter picks the scheme for each account and payee pair.

Scheme	When it is used	Why
exact	One-off payments, and every off-platform payee	Nothing to amortise. The payment is the settlement
upto	Metered calls where the final price is not known up front	Authorize a ceiling and settle the actual amount
batch-settlement	High-frequency calls to the same payee	800 payments of $0.02 become one on-chain transfer

A payee starts on exact and moves to batch-settlement once repeat volume justifies opening a channel. The agent never sees any of this.

Three records matter for reconciliation, and each has a different job. The internal ledger holds per-payment truth and is the system of record. Signed cumulative vouchers are the per-payment audit evidence. The on-chain batch proves net settlement and nothing finer. On-chain data is therefore not the audit trail. In a batch, the chain records who submitted the transaction, not the payer or the individual payments. We prove solvency with the hourly check that payables never exceed holdings.

Running batch settlement makes Cleargate the facilitator for its own channels, and that concentrates trust. A compromised facilitator could misdirect many payments at once. We treat voucher-signing authority as a production secret of the same class as the settlement wallet. No one can move funds alone, approvals are dual above set thresholds, and signed totals are verified independently.

Cross-chain payments use float and not bridges. If a payee wants a different chain from the one the buyer funded, the ledger debits and credits internally and the payout comes from float held on the payee's chain. Rebalancing between chains is an occasional treasury task, not part of any payment.

Cryptography

We use three things and nothing exotic.

- **Signed mandates.** The owner signs and the server verifies. A database breach cannot fabricate spending authority, and a dispute has a provable artifact.
- **Signed decision records.** Both parties can verify that a record was not altered afterwards.
- **A hash-chained audit log.** Each entry includes the hash of the one before it, so tampering with history is detectable.

Wallets, custody and gas

Key custody is bought, not built. Our initial provider is Coinbase Developer Platform (CDP). It is Base-native and comes from the same team as x402, and it bundles key custody in trusted execution environments, spend permissions, policies, gas sponsorship and address screening. It sits behind a wallet-provider interface, so it can be replaced. Wallets can be created from an email login with no seed phrase, which is what makes "no wallet, no chain, no gas" literally true.

Two control models serve two audiences. Server-controlled wallets give non-crypto users a bank-like experience, and they are custodial in the regulatory sense. User-controlled embedded wallets suit crypto-native users who want their own custody, and they delegate time-limited authority to our backend.

Where a provider policy can express a mandate rule, we set it there too. That puts part of the mandate below our own servers, so even a breach of our backend cannot exceed it. Our engine enforces what the provider cannot: intent, behaviour, category and provenance.

Provider gas sponsorship covers exact payments. In the netted flow agents mostly do not transact on chain at all, since only settlement does. We monitor and auto-refill the settlement wallet's gas balance, because a settlement failing on our own empty wallet is a preventable outage.

Funding, verification and withdrawal

Verification is tied to the spending limit, never to the funding method. A user who funds $20 by card should not face more friction than one who deposits $20,000 in stablecoin.

Tier	Requires	Unlocks
Basic	Email	A low cap for trial-sized spending
Verified	Identity verification	Standard limits
Business	Legal entity and beneficial ownership	High limits

The funding route then sets the unavoidable floor. Card and bank funding runs through an on-ramp, which is the regulated party and sets its own minimum checks. Direct stablecoin funding is on our terms, so a low-cap trial needs no formal KYC but does need on-chain address screening. Screening asks whether an address is tied to sanctioned or criminal activity, while KYC asks who the person is, and the two are not interchangeable.

Withdrawals follow the same logic. To a crypto wallet, we screen the address and keep friction low for small amounts. To a bank account, an off-ramp is involved and KYC is unavoidable. AML responsibility stays with us on both paths.

Platform security

Sitting in the payment path makes us a target, so we assume attackers will look at us too.

- Private keys are never held in-house. Custody is delegated and the keys live in trusted execution environments.
- No single person can move funds, and payouts above a threshold need dual approval.
- Secrets live in a managed vault and never in source control.
- Every action is logged with actor, action and timestamp, including internal administration.
- Rate limits apply per agent and per account.
- Alerts fire on spend spikes, new merchants with unusual volume, rising refund rates and ledger imbalance.

Engineering practice is plain by design: continuous integration on every commit, one-click deploy and rollback, feature flags, a trace ID through every request, load testing before launch and written incident runbooks.

8. Accounts, agents and money

A principal owns an account, the account contains users and agents, and each agent holds a mandate. For platforms the model nests: a platform sets a ceiling, its customers' sub-accounts sit inside that ceiling, and their agents sit inside those. A retail user is the same hierarchy with the upper layers collapsed, which means one person, one account and one implicit owner. The objects and the code path are identical, and multi-tenancy is built in from the first commit because retrofitting it is a rewrite.

Agents hold authority, not money

Giving each agent its own wallet or slice of money is the wrong design. We use the corporate-card model instead. The account holds one pooled balance, each agent's mandate carries limits, and spending draws from the pool.

There are three reasons. Pre-allocating per agent strands capital, because an idle agent's slice cannot serve a busy one. Per-agent wallets multiply the crypto problem the platform exists to hide, since each needs gas, funding and keys. And scattered balances make reconciliation miserable, while one pool and a ledger give a single number anyone can verify.

In practice, funding converts fiat to stablecoin once, at account level. Setting an agent's limit moves no money and only writes a rule. When an agent pays, the ledger debits the account and raises that agent's *spent counter*, a meter that enforces the cap and holds no funds. Authorization checks both levels: is

there money in the pool, and is the agent under its limits. Per-agent remaining limits can therefore add up to more than the account balance, just as three employees with card limits can collectively be allowed more than the company's account holds.

Creating an agent issues a credential. Revoking it cuts access instantly, with no funds to sweep back.

Roles

Four permission roles cover every case.

- **Owner:** everything, including billing and the commercial relationship.
- **Admin:** manages agents and mandates, but not billing.
- **Guardian:** stops things. A guardian can veto a pending change to spending authority or freeze the account, and can propose nothing.
- **Viewer:** read-only access to the console and reports.

Two separations do the work. *Propose versus stop* is the security boundary. On a guarded account no single compromised person can widen what an agent may spend, because a cap increase waits out a delay in which a second person with separate hardware can veto it. *Own versus operate* lets a business hire someone to run its agents without giving them the power to close the account or move its billing. A guardian's power is deliberately subtractive, so a stolen guardian credential is worth nothing on its own.

These controls are opt-in and scale with the customer. A retail account has one owner, no guardian and no delay. Changes apply at once and the owner is notified. A timelock adds a waiting period even with nobody to veto, which turns a stolen session from an instant loss into an emailed warning. A guardian adds a second pair of eyes. Enabling a protection is instant, while removing one is delayed and notified, because disabling the alarm is an attacker's first move.

Platforms and enterprises

Platforms embed Cleargate instead of using its console. Through the API they create sub-accounts, provision agents, apply mandates across a fleet, revoke access instantly and read decision history. Webhooks report events such as payment approved, held and declined, approval required, anomaly detected and delivery confirmed. Embeddable components, a mandate editor and a spend console under the platform's own brand, let platforms integrate instead of building their own policy interface.

Policy inheritance bounds a platform's risk whatever its end users do. The platform sets a ceiling, its customer sets policy inside it, and the agent operates inside that. A child can never widen what a parent set. Platforms can prefund and carry the loss, have customers fund their own sub-accounts, or have customers fund while the platform backstops up to a cap.

Enterprises ask a different question. A developer asks whether the API response arrived. An enterprise asks whether the agent spent in line with a mandate a human genuinely authorized, and whether that can be proved to an auditor. They need dual control above thresholds, policy as code within existing change management, and budget attribution by agent and cost centre that reconciles into ERP. They also need an immutable audit trail with SIEM export, separation of duties between the person who writes a policy and the person who approves an exception, and SOC 2 Type II. This is less about fraud detection and more about provable chains of authorization.

9. Agent-to-agent payments

Paying another agent is the same design with a different payee, but two distinct cases hide under that name.

Case A: an agent is the payee. A provider agent sells a service and a client agent pays for it. To the system this is identical to paying a merchant. The provider has an endpoint, quotes a price, is paid and delivers. The counterparty check, holds and delivery verification all apply unchanged. This case works with the core design today, because "merchant" is a role any account can take.

Case B: an agent subcontracts a job. A client agent hires a provider to do work, such as analysing a dataset, watching a feed or writing code. This differs in three ways. Delivery may not be a schema check, because the deliverable can be the subjective thing itself. The work takes minutes, hours or days, which breaks the short window of a simple hold. And delegation chains form, as a provider hires a sub-provider.

The foundations for Case B exist already: mandates, signed authorization, a delegation-depth field, a ledger where an agent-to-agent payment is just an entry between two accounts, and the decision log. Three pieces are not built yet.

- **Milestone holds,** which release payment in stages against checkpoints to limit the risk on long jobs.
- **A defined acceptance mechanism.** For work with a checkable result, such as passing tests or a matching schema, this can be automated. For subjective work it cannot, so we would facilitate those jobs and not guarantee them.
- **Chained mandate enforcement.** When one agent hires another, the sub-mandate must be strictly narrower than the hiring agent's authority, signed and depth-limited. Authority must never pass along implicitly.

We ship Case A with the core. We are keeping the primitives for Case B open, and we will build its machinery when multi-agent commerce is real and not only promised.

10. Discovery and the directory

x402 settles payments, but it does not help an agent find a payee worth paying or tell it whether that payee delivers. We fill that gap with a directory and trust registry. It is not a marketplace.

The directory lists HTTP endpoints, MCP servers and service-providing agents. Each entry has a description, a category, a price and verified statistics: delivery rate, transaction count and age. The statistics are the product. Anyone can describe themselves, but only a party in the payment path can report what happened afterwards.

Two listing tiers make the directory useful before anyone signs up.

Tier	How it gets there	What it shows
Registered	The payee claimed it	Full statistics and a verified delivery record

Tier	How it gets there	What it shows
Observed	Agents paid it and we recorded the outcome	Limited display, plus a link to claim the listing

We keep two facts apart: payment observed and delivery observed. We always know the first. We know the second only where delivery was confirmed or we proxied the request. A listing never implies quality we did not witness.

Open to read, gated for protection

Surface	Access
Read the directory and reputation	Open, rate-limited
Claim or edit a listing	Verified payee only
Mandate checks, holds, netted settlement and key safety	Registered agents only

Locking the directory to turn it into a signup incentive would be the wrong trade. A widely read registry pulls in both sides, since merchants come for reach and agents come for inbound. It also makes "Cleargate-verified" a trust standard that reaches beyond our own customers. A locked directory turns a compounding network effect into a modest signup incentive.

How an agent uses it

The directory is exposed to agents as an MCP server, so using it takes one line of configuration. A connected agent checks the directory first, and the owner's mandate can state a preference for verified payees, which biases selection without forcing it. If nothing fits, the agent goes to the open web and is still protected by every buyer-side check. The directory improves outcomes and does not gate them. An agent that can only buy from a registry is an agent whose owner is locked in.

11. The console and human control

The decision log is the product. It is tempting to assume the chain already shows what happened, but it does not. In the netted flow an individual agent payment never touches the chain and has no transaction hash. It is a ledger entry plus a signed voucher.

Almost everything a customer wants to see therefore exists only inside Cleargate: the individual payments, and the declines and holds that show the value we add. A monthly summary such as "blocked 14 payments, $3,200 at risk, 2 unknown merchants held for review" is the renewal argument, and the system produces that evidence as a byproduct.

What the console shows

- **Ledger events, not hashes.** Each row has an event ID, a time, a payee, a stated purpose, the mandate version in force and the outcome. Hashes appear in exactly three places: funding, withdrawal and batch settlement.

- **One hash per payout.** On the payee side, a payout carries one hash with its constituent payments listed beneath it, so a merchant can reconcile one on-chain receipt against hundreds of sales.
- **Two separate facts.** Payment observed and delivery observed are never merged into one status.
- **Every decision.** That includes decisions where no money moved, each with its reason.
- **The causal chain.** From payment, to the task that triggered it, to the agent's stated purpose. This makes a hijack visible in hindsight.
- **Spend views and exports.** Burn-down against daily and monthly caps, spend per merchant with delivery rates, attribution by customer, task or workflow, and export to CSV, accounting tools and SIEM formats.

Human control is set once

Control is configured in advance and enforced automatically. A mandate is set once and applies to thousands of payments without anyone being asked. Live human approval is the exception, kept for genuinely unusual events. An agent that needs a human for ordinary work is an agent nobody will deploy unattended.

On suspicion the default is to hold and notify, not to block and wait. Blocking strands an unattended agent mid-task. Holding lets the task continue while the money stays under control and a person is told.

Where approval is required, speed matters, because the agent is waiting. The flow is a push or Slack notification, a one-tap decision and a hard timeout. If approval takes ten minutes, customers raise their thresholds until the control means nothing, which is how security features usually die. On timeout the default is decline, and auto-approve has to be chosen deliberately.

Strictness is set per account. A crypto-native trading agent wants wide autonomy and few interruptions, while an enterprise wants four-eyes on anything unusual. Strict is the default. We audit every synchronous human step in an agent's path as a friction risk.

Beyond security: finance operations

These problems sell on an ordinary Tuesday and not only after an incident, and each one is a byproduct of data we already collect.

- **Spend attribution.** Companies running agents often cannot tell which customer, task or workflow caused which spend. That makes unit economics impossible for anyone reselling agent work.
- **Zombie subscriptions.** Agents sign up for recurring API access with no central view. Across hundreds of agents, charges pile up that nobody can list or cancel.
- **Receipts, invoices and tax.** Thousands of machine transactions arrive with no invoices, no VAT handling and no accounting export, which is a compliance problem in the EU.
- **Price sanity.** Agents do not comparison shop. They pay the quoted price. Knowing category rates lets us flag overpayment, which creates value and is an easier sale than loss prevention.
- **Refunds for honest error.** Normal commerce has returns. Agent commerce has nothing for the case where an agent simply buys the wrong thing.

12. Target customers

Our customer is whoever is financially liable when an agent gets it wrong. Today that means businesses that run agents. Consumers come much later.

Segment	Their problem	Why they matter to us	When
AI companies running agents that buy APIs, data and inference	No view of what their agents spend, and no way to tie that spend to a customer or task	The pain is real today. This is where we start	Now
Agent platforms and marketplaces	They have to offer payments to their users and will not build risk systems themselves	One integration reaches thousands of agents and builds our delivery record fastest	Now
API and data merchants	They cannot verify who is paying them, and they absorb every dispute	The supply side. Their deliveries make up the delivery record	After buyers are routing payments
Enterprises running internal agents	Pressure to govern agent spending and prove a human authorized it	The largest contracts	From 2027, once SOC 2 Type II is in place

A good first partner

The strongest early partner is an "internet of agents" platform, where non-technical people describe an agent in plain English and that agent buys capabilities from other agents. It concentrates what the product does best: agent-to-agent payments with the payee as a role (Section 9, Case A), users who must never see a wallet, and distribution to thousands of agents through a single integration. Cleargate sits underneath as the invisible payment and trust layer, through the API and embeddable components.

One thing to settle with any such partner is whether their agents make simple requests and get responses, which we support now, or take on open-ended subcontracting, which needs the later machinery described in Section 9.

What we are not going after yet

Physical goods are out of scope for now. Cards already serve a $100 pair of shoes well, the economics do not need sub-cent payments, and a machine cannot verify delivery. Without that, the hold, the delivery record and the counterparty score all lose their footing. We focus on digital services, endpoints, MCP servers and data, where x402 wins on cost and delivery can be checked. Physical commerce can follow later through a platform connector, never merchant by merchant.

Merchants: win buyers first

We do not buy merchants. Buyer-side control is useful with zero merchants integrated. The sequence runs like this.

1. **Win buyers.** Control, visibility and key safety work against any endpoint on the open web.
2. **Route their payments,** including to merchants who have never heard of us.
3. **Collect per-merchant data:** who is paid, how much, how often, and whether they delivered.
4. **Convert merchants with their own traffic.** The pitch is concrete: agents already paid you $X through Cleargate this quarter, so claim your listing, settle instantly and stop handling gas.

13. Business opportunities and where the product can grow

Payments are the way in and not the destination. Sitting in the payment path gives us two assets. One is the position: every machine payment is checked by us before it moves. The other is the data: a verified record of who delivered and who did not. Everything below builds on one or both, and the data is nearly free to collect from day one, so we start collecting it before the products that use it exist.

Four directions of growth

- **Down the financial stack.** Authorization first, then holding funds, then credit, then underwriting. Each step needs data from the one below. After enough observed transactions we can extend spending limits without prefunding, and once default can be priced we can sell coverage.
- **Across rails.** x402 first, then ACP, AP2 and card rails. Merchants are half the network, and card rails carry most merchant volume.
- **Across actors.** Software agents first, then robotics and industrial systems. The primitives stay the same: a mandate, a decision, a settlement and an audit trail.
- **Into compliance.** When regulators require proof of who authorized a machine to spend, every regulated company running agents will need the audit trail we already produce.

Products this position can support

Opportunity	What it is	What it depends on
Merchant onboarding	Making a business agent-ready in an afternoon: wrapping an existing endpoint, generating payment and product metadata and handling the protocol. It is also how we bring the supply side onto the platform	Nothing beyond the core product
Buyer-side intelligence	Price intelligence that flags overpayment against category rates, plus spend analytics and attribution for a company's agent unit economics	Data we already collect

Opportunity	What it is	What it depends on
Seller trust ratings	A displayable reputation score for every merchant, sold as an API to parties who route no payments through us, plus portable merchant identity. The record is observed, not self-reported, so it cannot be faked	Broad coverage of merchants. It is a year-three product that rests on a year-one data decision
Credit	Agent credit lines that spend against a limit instead of prefunding, merchant advances priced off reputation, and buyer protection sold to platforms that route over other rails	Default that can be priced from delivery history. Structured through an MGA, and the endgame that carries the most capital and regulatory weight
Compliance tooling	Proof that a human authorized a machine to spend, for regulated companies	The regulator's timeline, so we treat it as upside and not a commitment
Agent-to-agent subcontracting	Paying for jobs and not just calls, with milestone holds, an acceptance mechanism and chained mandates (Section 9, Case B)	Multi-agent commerce becoming real. The qualified seller base is still in the low thousands
Agents acting in DeFi	The engine authorizes movements of value against a signed policy, and does not assume a payment is a purchase. An agent in DeFi needs the same engine with different checks: approval caps, transaction simulation, slippage bounds and protocol allowlists	A deliberate decision to build a second product. It is not being built now

The order matters. We collect the data for the trust ratings, the analytics and price intelligence from the first transaction, because that is cheap and it compounds. We build the products that depend on scale, such as the reputation API, credit and guarantees, only once that scale makes them credible. Compliance tooling and DeFi stay deliberate later bets, since chasing them now would split our focus.

The end state

The end state has two engines working together. One is an authorization network, where every machine payment above a trivial size is checked before it runs. The other is a trust ratings service. It keeps the

record of who delivered and who defaulted across the machine economy, much like a credit score for sellers, and licenses it at almost no marginal cost to parties who never route a payment through us.

14. Risks

The most likely way this fails is that agent commerce grows more slowly than expected. The table lists that and the other risks we manage.

Risk	Why it matters	How we handle it
The market stays small through 2028	The most likely failure mode	Subscription and visibility revenue that does not depend on transaction volume
Incumbents ship basic controls as a feature	Wallet and CDN providers may add spending limits	They are unlikely to underwrite risk. We compete on loss data and, later, guarantees, and not on rules
Underwriting losses exceed capacity	Could end the company in a single quarter	Conservative limits, guaranteeing only what we could lose entirely, and an MGA structure before scaling
False positives drive customers away	Blocking a legitimate agent can cost more than the fraud prevented	Hold-and-notify by default, and a published false-positive rate
Custody and licensing	Server-controlled wallets are custodial in the regulatory sense	A user-controlled option alongside, and regulatory advice before launch in each market
Facilitator compromise	Batch settlement concentrates authority across every open channel	Voucher-signing keys treated as production secrets, dual approval and independent verification of signed totals
Overclaiming verification	Promising judgement on subjective quality creates legal exposure under a guarantee	Machine-checkable conditions only, stated explicitly in contracts
Platform compromise	We are a high-value target in the payment path	External key management, no unilateral fund movement and independently verifiable signed mandates

Risk	Why it matters	How we handle it
Availability	Downtime stops customers' agents	A four-nines target, a real on-call rota and an honest status page
Two-sided cold start	Merchants need agents and agents need merchants	Largely avoided, because buyer-side control works with zero merchants and routed traffic converts merchants
A crowded field	Wallets with spend controls are a contested category	We position on the combination described in Section 3 and not on "agent wallet with limits"

15. Glossary and references

Glossary

Term	Meaning
Principal	The person or legal entity ultimately responsible for an account
Account	The container for a balance, users, agents and services. Neither buyer nor seller by nature
Agent	The autonomous actor connected to an account that initiates payments under a mandate
Mandate	A signed grant of spending authority to an agent, enforced below the model and versioned
Counterparty	Any party an agent pays. Reputation attaches here
Payer and payee	Roles taken per payment. "Payee" is what was historically called the merchant, and any account can be one
Spent counter	A per-agent meter that enforces a mandate limit. It holds no money and resets each period
Decision record	The immutable, signed log entry for an evaluated payment, whether approved, held or declined
Hold	A conditional settlement released on confirmed delivery and refunded automatically on expiry

Term	Meaning
Netting	Combining many payments to one payee into a single on-chain settlement
Scheme	The x402 settlement mechanism in use: exact, upto or batch-settlement
Voucher	A signed off-chain message stating the cumulative total owed on a payment channel. Each one supersedes the last
Channel	The on-chain escrow and voucher series for one account and payee pair
Facilitator	The party that redeems vouchers on chain
Loss observation	An internal record of what would have been a covered loss under a guarantee. Measured from the first transaction and paid to no one
Rolling reserve	A trailing percentage of a payee's payouts retained to fund refunds
MGA	Managing general agent. Prices and writes risk while a reinsurer holds the capital
Idempotency key	A client-supplied identifier that stops a retried request from paying twice
Directory	The readable trust registry of payees and their verified delivery statistics. Not a marketplace

References

- Sam Boboev, Deep Dive: The hidden liability of agentic commerce, Finextra. Why the merchant is left holding the cost when an agent buys the wrong thing.
- Chargeflow, Agentic Commerce Regulation 2026: What Merchants Must Know. How Regulation E and the EU AI Act apply in part, with gaps around authorization and liability.
- Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents, ICLR 2025. The 84% average attack success across 13 model backbones.
- Qiusi Zhan, Richard Fang, Henil Shalin Panchal and Daniel Kang, Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents, NAACL Findings 2025. Eight defences, all bypassed.
- SlowMist, Behind the Grok Exploitation: An Analysis of AI Agent Permission Chain Abuse. Forensic account of the May 2026 incident.
- OECD AI Incidents Monitor, AI Prompt Injection Exploit Drains Grok-Linked Crypto Wallet.

- Chainalysis, Inside x402: 100M Agentic Payments on Base. Transaction-size trends and volume composition.
- Lucas Shin, Artemis Research, Machine Economy 2030. Supply-side and discovery analysis.
- Forkast, Cloudflare Just Gave AI Agents a Budget. Now the Agents Can Finally Pay. Source of the 14% consumer trust figure, which comes from a Product.ai report.
- x402.org. The open protocol specification.

This paper describes our current design and plans. Roadmap items and future products are intentions and not commitments. Keep in mind that the product is in active development and whitepaper might change.