TrustedPAI

Technical Protocols Documentation

Enrico De Vito - enxdev / 0xENX

Introduction: why this document exists, and how to read it

This document explains what TrustedPAI is, why it exists, and how every single piece of it works (protocols, risk engine, features, security, architecture), with diagrams, step-by-step practical examples, frequently asked questions, and the real, honest state of the project.

This document grows out of a request that sounds simple on paper but is actually quite ambitious: explain TrustedPAI to someone starting from zero, but explain it all the way through, not a summary for insiders, not a sales brochure, but something you can read from beginning to end and, by the last page, have understood every piece: what the project does, why it exists, how each of the protocols it deals with actually works internally, what really happens when something goes wrong, and, just as important, what it still doesn't do today, and why.

It isn't a technical document in the strict sense (it isn't meant for someone who needs to write code against the API), and it isn't a purely popular-science document that stays on the surface either. It's something in between: every technical concept is first introduced with a simple analogy, often paired with a diagram, and only afterward explained in real depth, with the actual names of the standards, the actual names of the fields, and the actual numbers behind the thresholds, so that the reader never has to choose between “understanding the gist” and “really understanding.”

The structure follows a deliberate path, meant to be read in order the first time, and then consulted in pieces afterward. It starts with the “what it is” in one sentence, widens out to the problem the project solves, takes a step back to understand where all of this comes from (the historical context), then moves into the actual mechanism, step by step, protocol by protocol, with concrete examples from start to finish, before moving on to the features built around the core, to security, to the questions anyone would naturally ask after reading all of this for the first time, and finally to the honest state of the project: what exists today, what's missing, and why it's deliberately missing.

The four points that are hardest to grasp from words alone (the complete flow of a transaction, the structure of AP2's three mandates, the logic behind reaching a verdict, and how the technical pieces connect to one another) are each accompanied by a diagram as well as text: if a sentence isn't quite clear on first read, the nearby image often makes it click immediately.

One last note before starting: agentic commerce, that is, payments made by AI agents on behalf of people, is a field that was literally born in 2025 and is still evolving rapidly as these lines are being written, in mid-2026. Some details in here (beta versions of protocols, partner counts, adoption status) will change in the coming months. What stays stable, and is the real heart of this document, is WHY all of this exists and HOW a system like TrustedPAI reasons when faced with a payment made by a machine: that underlying logic changes far more slowly than the individual technical details.

At the back of the document you'll also find a complete alphabetical glossary and a frequently-asked-questions section: if a term isn't clear while you're reading, or if a question occurs to you before you even reach it in the text, those two sections are meant to be consulted out of order too.

1. What TrustedPAI is, in one sentence, and then in full

TrustedPAI is a service that checks, BEFORE a payment is executed, whether an AI “agent” buying something on behalf of a person truly has permission to do so, and whether that transaction looks risky or suspicious.

In practice it's a gatekeeper: it never moves money, but it issues a verdict (go ahead, needs a human look, or block) before the payment goes through. The best comparison is a bouncer at the door of a club: they check IDs and decide who gets in, but they never touch the cash register.

Put a bit more technically, but still in plain language: TrustedPAI is an API (that is, a service other programs connect to over the internet to exchange information, without human involvement) for risk-screening specialized in agentic commerce: that slice of online payments where the one who “clicks buy” is no longer necessarily a human in front of a screen, but an AI program acting on their behalf.

A natural question at this point: why is a dedicated service needed, when the fraud-prevention systems an online store already uses today for normal human payments should be enough? The short answer, expanded on in section 2, is that traditional fraud-prevention systems are built around human behavioral signals (typing speed, mouse movement, time spent on a page, browsing patterns), signals an AI agent simply doesn't generate, or generates in a completely different way. A system built to spot a suspicious human isn't automatically good at spotting a suspicious AI agent: they're related problems, but different ones.

2. The problem it solves: what “agentic commerce” is

Until recently, every online payment started with a human gesture: a real person clicking “buy” on a website, after looking at the price and the product. That click was, in effect, the proof that someone had genuinely decided to buy, at that moment, at that price.

Today that's increasingly no longer the case. An AI assistant (ChatGPT, Gemini, or an agent custom-built by a company) can buy something on a person's behalf, sometimes while that person isn't even looking at a screen. Two concrete examples, quite different from each other:

> *The “present” scenario: you ask an assistant “find me some white running shoes under €150,” the assistant shows you a cart with two pairs, you look at it and confirm right there. The human is present and approves at the exact moment of purchase.*
>
> *The “delegated” scenario: you say, just once, “buy me a ticket for the concert as soon as it goes on sale, no more than €100, only if it's on a weekend,” and the agent carries out the actual purchase weeks later, on its own, with nobody watching at that precise moment.*

The second scenario is the new one, and it creates a problem that didn't exist before: how does the store know that agent REALLY has that real person's permission, for THAT specific purchase? How does it notice if the agent made a mistake, if someone tampered with the order along the way, or if it's deliberate abuse: a compromised agent, or an ambiguous instruction misread? Before, the human click was itself a minimal guarantee against all of this. Now that check can be entirely absent, so something else is needed in its place. This is exactly the space TrustedPAI operates in, and the reason the five protocols described in section 5 came into being.

It's worth being concrete about what can actually go wrong, because these are very different risk categories that often get lumped together:

- **Good-faith agent error**: the AI misreads an ambiguous instruction (“buy something nice for my daughter's birthday” leaves too much room for interpretation) and purchases something the person wouldn't have chosen. It isn't an attack, it's a misunderstanding.

- **Compromised agent**: someone manages to manipulate the agent from the outside (for instance through hostile content hidden in a web page the agent reads) to get it to make a purchase the user never asked for.
- **Tampering in transit**: the order starts out correct, but someone between the agent and the store alters the price or the recipient before the payment is executed.
- **Theft or reuse of an authorization**: a legitimate payment authorization, already used once, is intercepted and replayed to pay a second time (the “replay attack” from the glossary).
- **Deliberate abuse**: someone builds an agent specifically to exploit loopholes in stores (for instance buying up a limited-stock deal in bulk, or generating fake orders to drain promotional credits).

Each of these categories calls for a slightly different check, which is why TrustedPAI doesn't run just ONE verification, but a combination: first cryptography (to make sure the authorization is genuine and hasn't been tampered with, covering categories 2, 3 and 4), then the rules-and-history-based risk engine (to catch suspicious patterns even when the signature is perfectly valid, covering categories 1 and 5).

3. A bit of history: how we got here

To understand why nearly five different protocols for letting AI agents pay were born almost simultaneously in 2025-2026, it helps to briefly look at where all of this comes from: it isn't a bolt from the blue, it's the (provisional) end point of a series of steps that unfolded over roughly thirty years.

From hand-typed card numbers to “click to buy”

In the '90s and 2000s, buying online meant typing your credit card number by hand, every single time, into a web form. It was slow, it was risky (the card number traveled around, in the clear or close to it, across different sites), and every site had to build its own payment system from scratch. Little by little, intermediaries like PayPal and later Stripe arrived, turning payment into “a service” a site could integrate without having to directly handle sensitive card data, and this is already a direct parallel to what TrustedPAI does today for risk: taking a delicate, specialized responsibility and offering it as a service to anyone who needs it, instead of having every single company reinvent it.

The fight against fraud, and the birth of “human presence” as proof

As e-commerce grew, stolen-card fraud grew right alongside it. The industry's response was to build increasingly sophisticated layers of identity verification: the security code on the back of the card (CVV), address verification (AVS), and then so-called “3-D Secure” (the thing where, paying online, a code sometimes arrives by SMS or a bank-app notification to confirm). All of these systems share one assumption: that there's a real person behind the screen, right at that moment, able to look at a message and respond. It's exactly that assumption that stops holding up once the one “clicking buy” is a program.

The API economy, and payments becoming programmable

Over the following decade, companies like Stripe made payments “programmable”: no longer just a form to fill out, but a set of functions a developer can call directly from their own code. This paved the way for everything we now take for granted: automatic subscriptions, marketplaces splitting payments across multiple parties, checkout embedded inside third-party apps. Without that being the stated goal, it also laid the technical groundwork agentic commerce rests on today: if a payment can be triggered by code written by a human, sooner or later someone would arrange for it to be triggered by code written (or run) by an artificial intelligence.

The arrival of AI agents capable of acting, not just answering

Between 2023 and 2025, large language models (LLMs, like GPT or Gemini) moved from simply “answering a question” to being able to use external tools, browse websites, and carry out sequences of autonomous actions to reach a goal set by a human, what the industry calls “agentic AI.” The moment these agents started being able to fill out forms, click buttons, and complete web processes on their own, buying something online became one of the first practical, economically significant applications, and with it came, almost immediately, the problem described in section 2: how do you know a payment made by an agent is truly authorized?

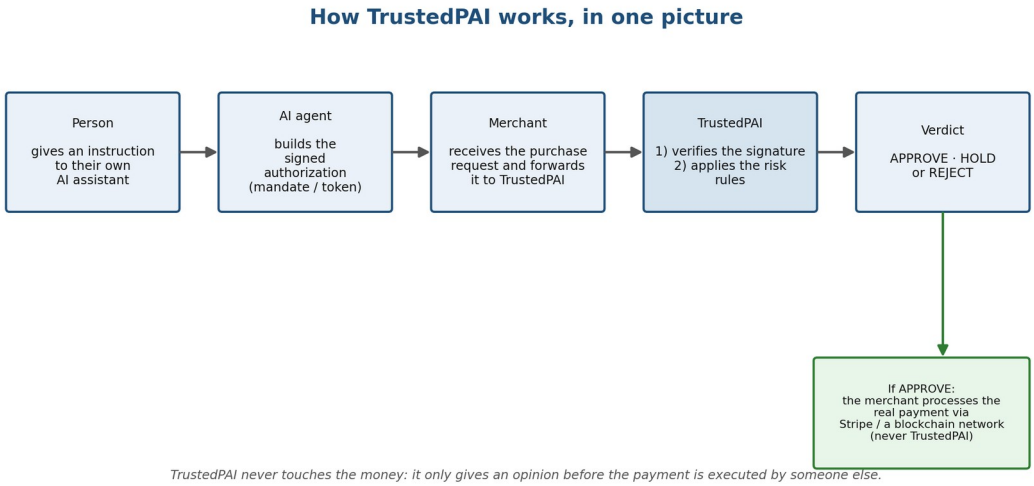
2025-2026: the race to build protocols

Over the course of a little more than a year, almost in parallel, several major industry players published their own protocols to answer exactly that question, each building on their own existing strengths: Google (with AP2) brought its experience in digital identity and verifiable credentials; Stripe and OpenAI (with ACP) brought payment infrastructure already used by millions of stores and hands-on experience with “Buy it in ChatGPT”; Coinbase (with x402) brought its experience in stablecoins and blockchain for instant, very-low-cost payments; Stripe and Tempo (with MPP, which arrived later, in March 2026) tackled the specific case of rapid-fire micropayments. It's a moment somewhat similar to the early years of the internet, when several competing protocols coexisted before one or a few became de facto standards, with the difference that, this time, the major companies involved started out collaborating on some pieces from the beginning (for example the declared compatibility between UCP and AP2) instead of ignoring each other.

It's in this precise historical moment, a very young industry, with more standards than are settled yet, where no one has definitively “won,” that TrustedPAI chose its position: not to bet on a single protocol, but to be neutral and support them all, so as to stay useful regardless of which standard ends up prevailing, or whether several specialized standards keep coexisting for different use cases (the more likely outcome, given how specialized each one is: AP2 for complex mandates, x402 for instant micropayments, MPP for continuous sessions).

4. How TrustedPAI works in practice, step by step

Let's follow a concrete example, start to finish, using the “delegated agent” scenario from before. First the bird's-eye view, then the step-by-step detail:



The full path: from the person's request to TrustedPAI's verdict, through to the actual payment.

1. When the moment comes to buy the concert ticket, the AI agent presents itself to the store (the “merchant”) with its own authorization, a “mandate” (in plain terms: a written permission, with a digital signature impossible to forge, proving the user genuinely authorized that purchase) in one of the formats described in section 5 (AP2, ACP, x402 or MPP).
2. Instead of processing the payment right away, the store sends the transaction details to TrustedPAI: a single API call, containing the mandate/authorization, the amount, and the agent's identifier.
3. TrustedPAI does two things, always in this order: first it verifies the authorization's cryptographic signature (to make sure it isn't fake, hasn't been altered, and hasn't already been used before); only if that check passes does it then apply the risk rules configured by the store (see section 8).
4. TrustedPAI responds with a verdict, usually within milliseconds: APPROVE (go ahead), HOLD (suspicious, needs a person to check before proceeding), or REJECT (block, do not proceed), along with a risk score from 0 to 100 and a text explanation of why.
5. Only at that point does the store decide whether to finalize the actual payment, which always happens through a real payment processor (Stripe, a blockchain network, etc.), never through TrustedPAI itself.

Key point to keep in mind for the rest of this document: TrustedPAI never touches money, at any stage of this process.

Who pays to use TrustedPAI? The store, or the PSP that handles payments on the store's behalf (such as Stripe). The owner of the AI agent (OpenAI, Google…) doesn't pay, and neither does the payment infrastructure itself (Stripe, the blockchain networks): TrustedPAI's customer is always whoever RECEIVES the payment, never whoever generates it.

An important point of philosophy, about what happens when something goes wrong INSIDE TrustedPAI (not in the transaction, in the service itself: e.g. the database is momentarily unreachable): the system never responds with a hard error. It still responds with 200 (technical success) and an explicit HOLD verdict, with the reasoning “automated evaluation failed, held for safety, requires manual review.” The idea is that the store always has something to act on (stop and check by hand), instead of being left with no answer at all or, worse, proceeding blindly.

5. Agentic payment protocols: what they are, one by one, in detail

An agentic payment protocol is a shared set of technical rules that establish EXACTLY how an AI agent must prove it has permission to pay, in a format any store, bank, or platform can understand and verify, without each of them having to invent its own system from scratch.

The industry was born in 2025 and is still very young: there is not yet a SINGLE universal standard accepted by everyone. Instead, several protocols have emerged, pushed by different companies, each with its own philosophy and its own backers. TrustedPAI today supports 4 of the 5 relevant ones, precisely so it can talk to any agent regardless of which technical "language" it speaks; it's a neutral role, not tied to any one protocol in particular.

Protocol	Who created it	What it solves, in brief	Status in TrustedPAI
AP2	Google + 60 partner organizations	Signed digital mandates, cryptographic proof of user permission	Stable
ACP	Stripe, OpenAI, Meta	"Buy it in ChatGPT": shared payment token	Stable
x402	Coinbase / x402 Foundation	Instant stablecoin payments inside a web request	Beta
MPP	Stripe + Tempo	Pre-authorized budget for rapid-fire micropayments in a session	Beta
UCP	Google + Shopify + 20 partners	Store/catalog/cart discovery, not payment in the strict sense	Not implemented

5.1 AP2: Agent Payments Protocol (Google)

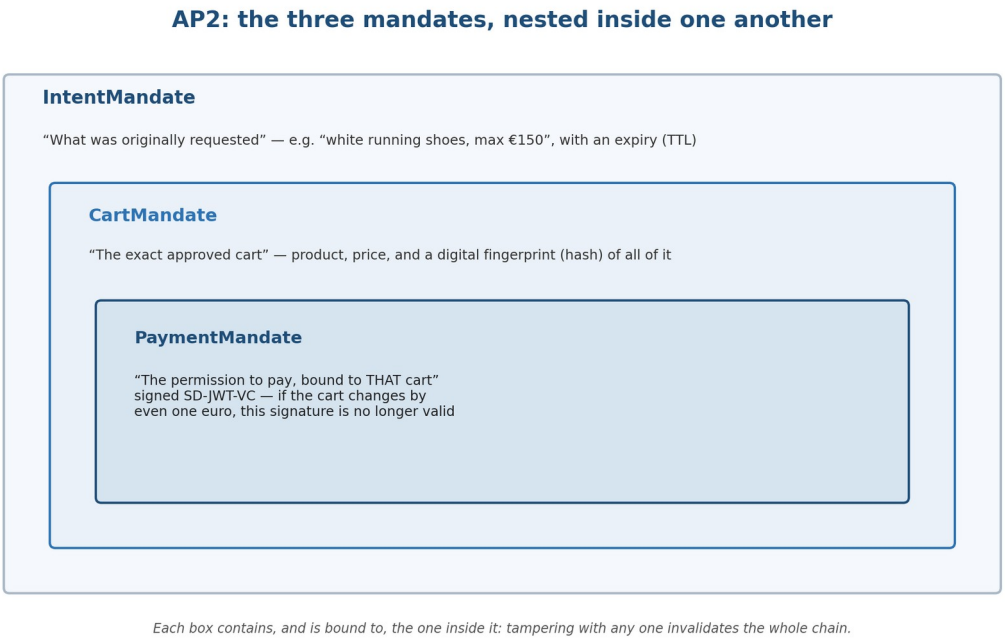
> **In short, in one sentence:** a system of three written, signed permissions, nested inside one another, that prove a real user genuinely authorized that exact purchase, at that exact price.

Who created it: Google, together with over 60 partner organizations, including PayPal, Mastercard, American Express, Coinbase, Etsy, Salesforce, Revolut, Adyen, ServiceNow, Intuit, MetaMask.

The problem it solves: before AP2 there was no standard way for an agent to prove “the real user genuinely gave me permission to buy exactly this, at this price.” Every company would have had to invent its own system, incompatible with everyone else's.

The three mandates, in detail

A “mandate” in AP2 is a cryptographically signed digital contract, impossible to forge or modify without the signature ceasing to be valid. There are three of them, nested inside each other like Russian dolls:



AP2's three mandates: each one bound to the one nested inside it.

Using the running-shoes example:

- **IntentMandate**: records what the user originally asked for (“find me white running shoes, no more than €150”). It serves as a verifiable trace of the initial request, and is especially important in the “delegated” scenario: it also contains a time limit (TTL, “time to live”) beyond which the authorization automatically expires, so it can't stay valid forever.
- **CartMandate**: the exact cart (products, quantities, total price) the user approved, or that the store proposed and signed to then be approved by the

agent according to the IntentMandate's rules. It contains a “hash” (a mathematical fingerprint) of the entire contents of the cart: if even a single number in the price changed after signing, the fingerprint would no longer match, and the tampering would surface immediately.

- **PaymentMandate**: links the approved cart to the actual payment method. It's the most delicate piece from a cryptographic standpoint: it uses a technology called SD-JWT-VC (a “verifiable digital credential,” a standard still being formally finalized, draft-ietf-oauth-sd-jwt-vc), extended with a field that binds the authorization inseparably to THAT specific CartMandate. In practice: even if someone stole this authorization, they couldn't reuse it to pay for a different cart; it's stitched onto that one transaction and nothing else.

How TrustedPAI actually verifies it

> ***You can skip this block if you're not interested in the cryptographic detail:*** *the next points explain EXACTLY how TrustedPAI checks the signatures, useful if you want to understand the mechanism all the way through, not essential if it's enough to know that “signatures are thoroughly checked, and attempts to cheat are caught” (you can pick back up at “Worked example” a bit further down).*

- It checks the CartMandate's signature (a JWT standard, ES256 format) against the store's public key, which is registered once in advance during onboarding and never exchanged inside the individual payment request.
- It checks the PaymentMandate's SD-JWT-VC signature against the public key of the entity that issued that credential (the user's trusted “credentials provider”).
- It recalculates the cart's fingerprint and compares it against the signed one, to catch any attempt to tamper with the price after signing.
- It keeps track of every payment identifier already used, to block replay attacks (see glossary).

There is, as of today, no public “test environment” for AP2 run by Google, unlike what Stripe offers for ACP (see below). Because of this, TrustedPAI internally generates test data that conforms to the official specification, with its own keys, so it can genuinely exercise the entire signature mechanism without depending on an external service that doesn't exist yet.

Status in TrustedPAI: stable. Verified with real tests against 7 different attack scenarios (amount tampered with after signing, authorization reused on a different cart, wrong nonce, fake keys, replay), all correctly blocked.

> ***Technical deep-dive (optional, advanced level):*** *governance of AP2 has moved from Google alone to the FIDO Alliance, the same consortium behind passwordless authentication standards ("passkeys"): a signal that the industry treats it as an open protocol, not the property of a single company. There remains, though, a real architectural limitation, shared across the whole AP2 ecosystem and not specific to TrustedPAI: today each client can trust only one "issuer" (the entity that issues the payment credential) at a time; a true multi-issuer trust registry doesn't yet exist. Independent research into AP2's security confirms this is a gap the industry as a whole recognizes, and suggests that a future evolution could correlate anomalies ACROSS different agents and stores, not just a single agent's own history, to uncover coordinated fraud patterns (so-called "slow cartel attacks," where several different agents share the same credentials provider or show inconsistent geographic patterns).*

Worked example: the running shoes, step by step

To make all of this concrete, let's follow the shoe scenario through to the end. The user tells their assistant: "find me white running shoes, no more than €150." Here's what happens behind the scenes, in simplified form (this isn't real code, it's a readable representation of the concepts):

```
IntentMandate:
  original_request: "white running shoes, max EUR 150"
  user: (signed user identifier)
  expires_at: 2026-08-25T00:00:00Z

CartMandate (after the agent found a pair at EUR 129):
  product: "Running shoes, model X, size 42, white"
  total_price: 129.00 EUR
  cart_fingerprint: 7f3a9c... (hash of everything above)
  signature: (ES256, store's key)

PaymentMandate:
  linked_to: cart_fingerprint 7f3a9c...
  payment_method: (user's tokenized credential)
  signature: SD-JWT-VC (credential issuer's key)
```

At this point the store sends all of this to TrustedPAI (endpoint POST /v1/screen-transaction). TrustedPAI first verifies both signatures and the cart fingerprint: if

even the price had been altered after the CartMandate was signed, the recalculated fingerprint wouldn't match and the transaction would be blocked instantly, before the risk rules are even consulted. If the signatures are valid, TrustedPAI then applies the store's policy (for example: “under €200, no history on this agent, automatic go-ahead”) and returns APPROVE with a low risk score, along with a short explanation. Only then does the store process the actual payment through its own processor, and the order goes through.

5.2 ACP: Agentic Commerce Protocol (Stripe, OpenAI, Meta)

> **In short, in one sentence:** a single-use payment “voucher” that lets the agent pay through Stripe without ever seeing the real card number.

Who created it: born out of a collaboration between Stripe and OpenAI: it's the protocol behind “Buy it in ChatGPT” (Instant Checkout), later broadened with Meta as a co-author of the open standard.

The problem it solves: the same need as AP2, but with an approach designed to integrate quickly with already-existing payment infrastructure (Stripe's above all, which already processes payments for millions of stores).

The "pieces" that make up ACP

ACP isn't a single mechanism, but a set of components that can be combined:

- **Agentic checkout**: the actual purchase session: creating, updating, and completing the cart, delivery options, payment processing.
- **Cart & feed**: a standard way for an agent to browse a store's product catalog and build a cart even before reaching payment.
- **Delegate payment**: the piece TrustedPAI focuses on: securely passing a “payment token” (Shared Payment Token, SPT) between the buyer, the agent, and the store, without ever exposing real card data to the agent. It's like handing someone a voucher spendable exactly once, for a precise amount, instead of your credit card number.
- **Delegate authentication**: delegating permissions via OAuth 2.0 (a widely used standard, the same one behind “sign in with Google” on countless sites), so the agent can act on the user's behalf at a store in a way that's traceable and revocable.

- **Orders & webhooks**: automatic notifications about order status after payment (confirmation, shipping, delivery, any refund).

How TrustedPAI uses it: it verifies the Shared Payment Token against Stripe's real sandbox (test environment), so not against made-up data, but against Stripe's actual test infrastructure. Status: stable.

Worked example: buying inside a chat

Imagine a user asking an AI assistant embedded in a chat app “buy this backpack you just showed me.” The assistant, via ACP, already has a Shared Payment Token in hand, generated earlier, when the user linked their payment method to the app, and valid for a single use up to an agreed maximum amount. The assistant presents this token to the store along with the cart (a backpack, €79). The store forwards everything to TrustedPAI (endpoint POST /v1/screen-transaction-acp), which verifies the token against Stripe's infrastructure: is it genuine? Hasn't it already been used? Is it within the agreed limit? If everything checks out, and the store's policy allows it, TrustedPAI responds APPROVE and the purchase goes through: the user never had to re-enter card details, and the agent never saw them.

5.3 x402 (Coinbase / x402 Foundation)

> **In short, in one sentence:** the site asks for payment (using an old web status code that had never really been used, “402”), the agent pays instantly in stable cryptocurrency, and only then receives what it asked for.

Who created it: Coinbase, now handed off to a broader foundation (x402 Foundation), with support from players such as AWS, Anthropic and Circle (the company behind the USDC stablecoin).

The problem it solves: instant stablecoin payments directly inside an ordinary web request, with no credit cards, no account creation, none of the friction typical of traditional online payments: designed specifically for very small, very frequent transactions, where credit card fees would make the operation uneconomical.

The HTTP 402 flow, step by step

x402 reuses an old, forgotten corner of the internet: the HTTP error code “402 Payment Required,” which has existed in the web standard since the '90s but had never actually been put to use until now. The flow goes like this:

```
1. The client (an AI agent) requests a paid resource from a
server
2. The server responds: 402 Payment Required + price and
accepted currency
3. The client signs a payment instruction and attaches it
to the request
   (in the HTTP header, not in a separate system)
4. The request is repeated, this time with the payment
included
5. A "facilitator" verifies the signature and settles the
payment on the blockchain
6. The server finally delivers the requested resource
```

> ***You can skip this block if you're not interested in the cryptographic detail:*** *the next paragraph goes into the actual signing mechanism (EIP-3009, EIP-712, ECDSA); if it's enough to know that "the payment is authorized with a secure digital signature and settled on a cheap blockchain network," you can skip ahead to the "How TrustedPAI uses it" paragraph.*

The signing mechanism: x402 uses the EIP-3009 standard ("transferWithAuthorization"), a function that allows authorizing a stablecoin transfer with an EIP-712/ECDSA signature (the same kind of cryptographic signature used by modern digital wallets) without having to pay a separate network fee just to "give permission." The payment is then settled on a "layer 2" blockchain (a network built on top of Ethereum to be much faster and cheaper, such as Base), a technical detail that explains why x402 has only recently become practical: on "vanilla" Ethereum, network fees would make micropayments uneconomical; on layer-2 networks they cost a few cents.

How TrustedPAI uses it: it genuinely verifies the cryptographic signature using the official Python "x402" library (not a homegrown reimplementation), and also reads directly from the blockchain (via RPC, "Remote Procedure Call," a way to query a network node directly) to check two things the signature alone doesn't guarantee: that the payment hasn't already been used before (anti-replay) and that there really are sufficient funds.

Status: beta: covers only the "exact" scheme on Ethereum-compatible blockchain networks (Base, Base Sepolia, Polygon), not Solana, not "mainnet" Ethereum, not other announced-but-not-yet-covered networks.

> ***Technical deep-dive (optional, advanced level):*** *x402's governance has also moved from Coinbase to a broader consortium, now under the umbrella of the Linux Foundation, again a signal of openness toward the whole industry, not control by a single company. An interesting internal architectural note: when work began on building x402 support into TrustedPAI, the starting assumption was that it belonged to the same category as MPP, a "session-based" protocol, with an open budget spent over time. Actually implementing it disproved that assumption: x402 turned out to belong to the same family as AP2 and ACP (one signature authorizes exactly one payment, not a whole session); it's instead MPP, as seen in section 5.4, that behaves differently. It's a useful reminder: even when a protocol's official documentation seems clear, the only way to be sure how it actually behaves is to implement and test it, not to stop at reading the spec.*

Worked example: an agent paying to read an article

A typical x402 use case: a research AI agent tries to open a paywalled article to answer a user's question. The publisher's server responds 402 Payment Required, quoting "0.02 USDC on Base." The agent, which has a small wallet pre-loaded with funds by the user precisely for this purpose, signs an EIP-3009 authorization for that exact amount and repeats the request with the signature attached. Before the publisher delivers the article, the transaction passes through TrustedPAI (endpoint POST /v1/screen-transaction-x402), which checks the signature, verifies on Base that this authorization hasn't already been used, and checks that the wallet genuinely has sufficient funds. All of this happens in a fraction of a second, with no action required from the user at that moment; the only human involvement was, earlier, deciding how much to load onto the wallet and what it could be spent on.

5.4 MPP: Machine Payments Protocol (Stripe & Tempo)

> **In short, in one sentence:** instead of authorizing one payment at a time, the user authorizes a maximum budget for an entire session, and the agent spends within that budget without asking permission every single time.

Who created it: born in March 2026 out of a collaboration between Stripe and Tempo, a blockchain network built specifically for payments made by machines.

The problem it solves: the other three protocols (AP2, ACP, x402) authorize one payment at a time, each with its own signature. MPP instead addresses a different scenario: an agent that needs to make many micro-payments in quick

succession within the same context, for example a cent for every call to a service, dozens of times per minute. Requiring a separate signature for each one would be impractical: MPP has a maximum budget authorized in advance for a “session,” and the agent then spends within that budget without needing to request permission every single time.

The three modes

> ***You can skip this block if you're not interested in the cryptographic detail:*** *the three modes differ mostly in technical signing details (EIP-712 vouchers, JWE encryption); if you're only interested in the general logic (“a budget agreed in advance, spent in small steps”), you can skip ahead to “How TrustedPAI uses it” a bit further down.*

- **“stripe”**: conceptually identical to ACP: the same Shared Payment Token mechanism.
- **“tempo”**: usage-based payments on a dedicated blockchain network (Tempo), with a system of signed “vouchers” (EIP-712 standard, the same one x402 uses) that report a growing CUMULATIVE amount: each new voucher states the total spent so far in the session, not just the latest increment, so it's always possible to verify the total has never exceeded the agreed budget.
- **“card”**: an extension designed for traditional credit cards (proposed by Visa): the payment payload is encrypted (JWE format) so that, by the protocol's own design, ONLY the store's PSP can actually read it, not TrustedPAI, not anyone else. Because of this, in this mode TrustedPAI performs only structural checks (is the message in the right format? hasn't it already been used?) and always explicitly states that cryptographic verification isn't possible; this isn't a limitation of TrustedPAI, it's a design limitation of the protocol itself.

How TrustedPAI uses it: it implements all three modes. When cryptographic verification can't be certain (“card” mode), the calculated risk score automatically rises: a “clean” go-ahead is never returned in that case, precisely so as not to make a transaction look safer than it really is.

Status: beta: the “tempo” mode doesn't yet read the payment channel's state directly on the blockchain (there isn't yet a confirmed public RPC node for the Tempo network), so it relies on the most recent signed voucher rather than on an independent on-chain check.

> ***Technical deep-dive (optional, advanced level):*** *internally, TrustedPAI classifies each protocol into two architectural families: "single mandate" (one signature authorizes exactly one payment: AP2, ACP, x402, see the box at the end of section 5.3) and "streaming session" (a single authorization covers multiple micropayments over time). MPP is today the only protocol in the second family, but only partially, and it's worth being fully honest about that: each voucher is still evaluated as a standalone unit when it arrives; there isn't yet a true "session opening" with persistent channel accounting maintained by TrustedPAI from one voucher to the next. Budget control still works correctly (the cumulative amount declared in each voucher can never exceed what was agreed at session opening), but the architecture designed specifically for streaming sessions remains, as of today, only partially realized.*

Worked example: a micropayment session

An AI agent orchestrating several paid tools to complete a complex task (for example: fetching data, translating it, generating an image, all through different usage-based services) opens an MPP session with a maximum budget of €5 for the entire session. Every time the agent uses a service, a new voucher is generated, signed, stating the cumulative total spent so far (€0.10, then €0.35, then €0.80...). Before being accepted by the service receiving it, each voucher passes through TrustedPAI (endpoint POST /v1/screen-transaction-mpp), which verifies the signature and checks that the declared cumulative amount never exceeds the €5 budget agreed when the session opened: if a voucher declared a cumulative amount above the budget, it would be rejected immediately, even if the signature were perfectly valid.

5.5 UCP: Universal Commerce Protocol (Google, Shopify and others) - not yet implemented

> **In short, in one sentence:** it isn't a way to pay, it's a way for a store to "be found and understood" by an AI agent; to actually pay, it will still use one of the other four protocols.

Who created it: Google together with Shopify, with over 20 partners on board including Etsy, Walmart, Target, Stripe, Visa, Mastercard.

What it does, and why it's different from the other four: AP2, ACP, x402 and MPP deal ONLY with the final part: authorizing the payment. UCP instead covers the entire purchase journey: how a store gets "discovered" by AI agents (Discovery), how it presents its product catalog in a standard format (Catalog), how it handles the cart and checkout (Cart & Checkout), and how it handles

identity/tokenized payment with cryptographic proof of consent. It was created to solve what Google calls the “N×N problem”: without a shared standard, every store would have to integrate separately with every single AI platform (Google, OpenAI, etc.), multiplying the work. UCP is designed to be compatible with AP2 specifically on the payment side: UCP handles “the store,” AP2 handles “the checkout.”

Why it isn't in TrustedPAI yet: it's the only one of the 5 relevant protocols still missing today. Since it isn't a payment-authorization protocol in the strict sense (it's closer to “how an online store structures itself to be visited by an agent”), its direct relevance to a risk engine like TrustedPAI depends on how it evolves over the coming months; it stays on the list of things to evaluate, it hasn't been ruled out.

6. A complete example, from start to finish

The previous sections explained each piece separately. This section puts them back together into a single continuous story, to see how they REALLY behave in sequence, with a concrete and believable case.

Marco uses a generic AI assistant to which he's given access to a payment method, via AP2, with a monthly spending limit of €300 agreed in advance with the assistant itself. One day he says: “if the Milan-Barcelona flight drops below €90 in the next two weeks, book it, otherwise wait.”

6. The assistant records the request as an IntentMandate: destination, maximum price, two-week time window, corresponding TTL.
7. Ten days later, the assistant finds a flight at €84 and builds the CartMandate: airline, date, price, with the cart's digital fingerprint and the store's signature (in this case, the online travel agency).
8. The assistant generates the PaymentMandate, linked to that exact CartMandate, signed with Marco's payment credential (SD-JWT-VC).
9. Before confirming the booking, the travel agency calls TrustedPAI at POST /v1/screen-transaction with all three mandates and the amount (€84).
10. TrustedPAI verifies the signatures: the CartMandate is genuine and untampered (the recalculated fingerprint matches), the PaymentMandate is genuine and tied exactly to that cart, and neither identifier has been used before.
11. TrustedPAI applies the travel agency's policy: “under €500, agent with no prior history, human presence not required under €200”; €84 is well within these limits, and Marco's agent history (queryable at GET /v1/agents/{id}/history) has no negative record at all.
12. Verdict: APPROVE, very low risk score (say, 4 out of 100), reasoning: “valid signatures, amount within policy, no history on this agent.”
13. Having received the go-ahead, the travel agency processes the actual payment through its own processor and confirms the booking. Marco only gets the notification after the fact: “I booked the flight at €84 as you asked.”

Worth noting: throughout this entire flow, TrustedPAI never saw a card number, never moved a single euro, and reached a decision in a handful of milliseconds, less time than a human would need just to open the confirmation email.

7. Two edge cases told in full: a HOLD and a REJECT

Knowing that three possible verdicts exist is one thing; REALLY seeing what happens when a clean APPROVE doesn't come back is another. Here are two concrete scenarios, one for each case.

7.1 A HOLD case: high amount, no confirmation of human presence

Same Marco, same assistant, but this time the original request was vaguer: “if you find a good vacation package before summer, go ahead and book it.” Months later, the assistant finds a package for €1,450 and books it autonomously, assembling the three AP2 mandates exactly as in the previous example: the signatures are all perfectly valid, no tampering, no replay.

The store sends everything to TrustedPAI. The signatures pass the cryptographic check without any issue. But the store's policy says: “above €1,000, require explicit confirmation of human presence in the PaymentMandate, otherwise HOLD,” and in this case that field isn't present, because Marco's original request was genuinely delegated, with no confirmation at the moment of purchase.

Verdict: HOLD, moderate risk score (say, 58 out of 100), reasoning: “amount above the €1,000 threshold with no confirmation of human presence, requires manual review before proceeding.” At this point the store can choose to contact Marco for explicit confirmation before finalizing, or leave the booking pending until it makes a human decision. TrustedPAI didn't block the transaction outright: it flagged that an extra check is needed, without giving either an automatic yes or no.

7.2 A REJECT case: amount tampered with in transit

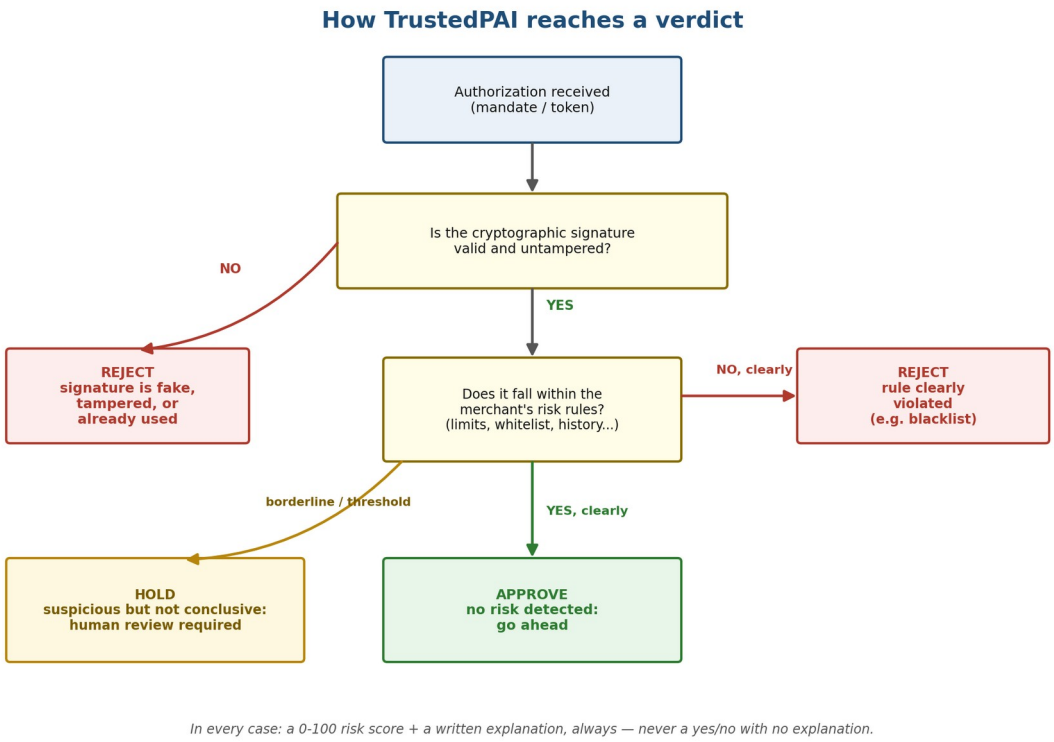
Another agent, belonging to another user, builds a CartMandate for a €45 electronics order, correctly signed by the store. Between the moment of signing and the moment the request actually reaches the store, something along the path (a compromised component, a tampering attempt) alters the total price field to €450, hoping the check will simply read the number without re-verifying it.

TrustedPAI, as it always does, recalculates the digital fingerprint of the entire cart from the data received and compares it against the one signed at the time the CartMandate was created. The two fingerprints no longer match: that's mathematical proof that something was altered after signing, whoever did it.

Verdict: REJECT, explicit reasoning: “cart fingerprint doesn't match the original signature, possible tampering, transaction blocked.” There's no room for ambiguity in a case like this: no HOLD is proposed for human review, because the cryptographic proof of tampering is already conclusive on its own; a human wouldn't have any way to “validate” a fingerprint that doesn't match anyway, so there's no point wasting anyone's time with a manual review in a case like this.

8. The risk engine: how TrustedPAI decides APPROVE / HOLD / REJECT

Verifying that a signature is genuine is only the first layer of checking, and it's a binary one (either it's valid, or it isn't). The second layer, independent of cryptography, is where TrustedPAI applies real risk judgment, based on the rules each individual store has configured for its own account. Every store has its own policy, different from any other store's; there's no single rule that applies to everyone. The diagram below sums up the whole logic at a glance; the paragraphs that follow explain it in detail.



From the cryptographic signature to the risk rules, through to the final verdict.

The risk dimensions evaluated

- **Maximum limit per single transaction**: e.g. “never auto-approve orders above €2,000”: beyond that threshold, the verdict tends toward HOLD even if the signature is perfect.
- **Maximum daily limit for the same agent**: to catch an agent buying in an unusual or excessive way within a short span of time, by summing all of its transactions for the day.

- **Whitelist / blacklist**: explicit lists of merchants/recipients always approved or always blocked, regardless of everything else.
- **Human-presence threshold**: above a certain amount, the store can require that it's been explicitly declared that a real person was present and confirmed the purchase at that moment, not just the agent on its own (especially relevant in the “delegated” scenario described in section 2, and seen in action in example 7.1).
- **Agent history and reputation**: if that same agent identifier already has confirmed abuse reports in the past (see section 10.3), the risk score calculated for subsequent transactions automatically rises: an agent “with a record” starts from a higher baseline of scrutiny.
- **Reliability of the cryptographic verification itself**: not all protocols/modes provide the same mathematical certainty (see MPP/“card” in section 5.4): when cryptographic verification can't be 100% complete, this counts toward higher risk, never the other way around.

The three possible verdicts

- **APPROVE**: no risk detected under the policy configured for that store: go ahead. Example seen in section 6.
- **HOLD**: suspicious but not conclusive: needs a person to check it by hand before proceeding. Typical examples: high amount with no confirmation of human presence (seen in section 7.1), agent with a history of risky prior activity, cryptographic verification not fully complete due to limits of the protocol/mode used.
- **REJECT**: blocks entirely: either the cryptographic signature doesn't check out (fake mandate, tampered, seen in section 7.2, or already used before), or the store's policy explicitly rules this case out (e.g. recipient on a blacklist).

> ***Technical deep-dive (optional, advanced level):*** *under the hood, every protocol connects to the risk engine through a shared software contract (an internal interface called PaymentProtocol), which exposes a single method, “verify and normalize,” capable of turning each protocol's specific format (AP2 mandates, ACP tokens, x402 signatures, MPP vouchers) into a common data structure the risk engine can judge without knowing which protocol it came from. This common structure carries an explicit boolean field, “cryptographically verified”: when it's false, for instance in MPP's “card” mode, structurally unverifiable (section 5.4), a dedicated risk-engine rule automatically*

> *raises the calculated score. This is the exact technical mechanism behind the claim, repeated several times in this document, that TrustedPAI never pretends to a cryptographic certainty it doesn't actually have.*

Every verdict comes with a numeric risk score from 0 to 100 and one or more text explanations of why; never a plain “yes” or “no” without reasoning, even when the verdict is APPROVE.

As already noted in section 4: if something goes wrong inside TrustedPAI itself (not in the transaction, in the service), the system never leaves the store without an answer: it still returns a cautious HOLD with explicit reasoning, never a silent error and never an automatic approval out of laziness during a moment of technical trouble.

9. The endpoints: what you can actually ask TrustedPAI

An “endpoint” is simply a specific API address a program can send a request to in order to get a certain service. Here's the complete list of what's available today, explained in plain terms:

Endpoint	What it does
POST /v1/screen-transaction	Evaluates a transaction authorized via AP2 (the Intent/Cart/PaymentMandate chain).
POST /v1/screen-transaction-acp	Evaluates a transaction authorized via ACP (Stripe's Shared Payment Token).
POST /v1/screen-transaction-x402	Evaluates a transaction authorized via x402 (on-chain stablecoin payment).
POST /v1/screen-transaction-mpp	Evaluates a transaction authorized via MPP, in one of the three modes (stripe/tempo/card).
GET /v1/health	Basic check: is the service reachable and working? Requires no authentication, useful before integrating the other endpoints.
GET /v1/agents/{id}/history	Returns the history and reputation of a specific agent: how many transactions, how many blocked, any confirmed abuse.
PATCH /v1/policy	Lets the store change its own risk rules on a self-service basis, without needing to request a manual change.
POST /v1/report-abuse	Lets the store report, after the fact, that a transaction that already went through turned out to be abuse (fraudulent return, suspicious refund, resale, unjustified chargeback, other).
GET /v1/my-transactions	Returns the calling store's transactions, with totals and outcomes, forming the basis of the customer dashboard.
GET /admin/dashboard	Internal monitoring panel (for whoever runs TrustedPAI, not for individual customers), protected by an access token: trend chart, transaction list, search filters.

All endpoints (except /v1/health) require an API key assigned to each individual customer, passed in the request header (“X-API-Key”). A missing, wrong, or deactivated key always returns an authentication error before the request body is even read.

Besides the classic REST route, all of these functions are also exposed through an MCP server (Model Context Protocol, a standard created by Anthropic; see section 10.5), so an AI agent can “call” TrustedPAI the way it would use a tool from its own toolbox, without going through a traditional web call.

10. Features beyond pure verification (the “competitive arsenal”)

Evaluating a single isolated transaction, one at a time, isn't enough to be competitive in this industry: that's proven by the fact that the main competitors (companies like Forter, Riskified, Signifyd, Visa) all offer something more than a simple point-in-time check. That's why TrustedPAI includes five additional features, all built by reusing the same risk engine, without duplicating logic:

10.1 Agent history and reputation over time

TrustedPAI doesn't judge only the single, isolated transaction: it keeps track of what that same agent has done in the past (total transactions, how many were blocked, any confirmed abuse) and uses that history to weigh the risk of subsequent transactions. One honest detail worth knowing: today this history is shared across ALL of TrustedPAI's customers for the same agent identifier, not isolated per individual store; if an agent has a prior incident with Store A, that incident also counts when the same agent shows up at Store B. It's effectively a first, still simple, form of shared reputation network: an agent that misbehaves doesn't “start clean” simply by switching stores.

10.2 Self-service policy builder

A store can change its own risk rules (spending limits, whitelist/blacklist, human-presence thresholds) on its own, independently, through the PATCH /v1/policy endpoint, without needing to request a manual change from the TrustedPAI team. The change genuinely applies to the risk engine in real time, not just to the confirmation response: the very next transaction, even seconds later, is already evaluated under the new rule.

10.3 Post-payment abuse management

A payment can look clean at the moment it's screened, and only turn out to be abuse afterward: a fraudulent return, a suspicious refund, an unauthorized resale, an unjustified chargeback. The store can explicitly report it to TrustedPAI via POST /v1/report-abuse, indicating the category; TrustedPAI factors this into future evaluations of the same agent (see 10.1). An important safety check: a store cannot report abuse on a transaction that isn't its own. The system always verifies this isolation, to prevent one store from damaging an agent's reputation over a transaction it never even handled.

10.4 Dashboard for the store

Through GET /v1/my-transactions, each store can see its own transactions (how many approved/blocked/under review) without having to read database rows by hand or request a manual report. This is separate from the internal admin dashboard (section 9), which is instead for whoever runs TrustedPAI itself and shows aggregated data across all customers.

10.5 An MCP server

MCP (Model Context Protocol) is an open standard created by Anthropic to let an AI assistant “talk” to external tools in a structured way, without having to build a custom integration every time. TrustedPAI exposes its own MCP server, making the same screening, agent-history-lookup, and abuse-reporting functions available via REST directly usable as “tools” by an AI agent, using exactly the same internal logic (no duplication, no risk of the two routes giving different answers for the same case).

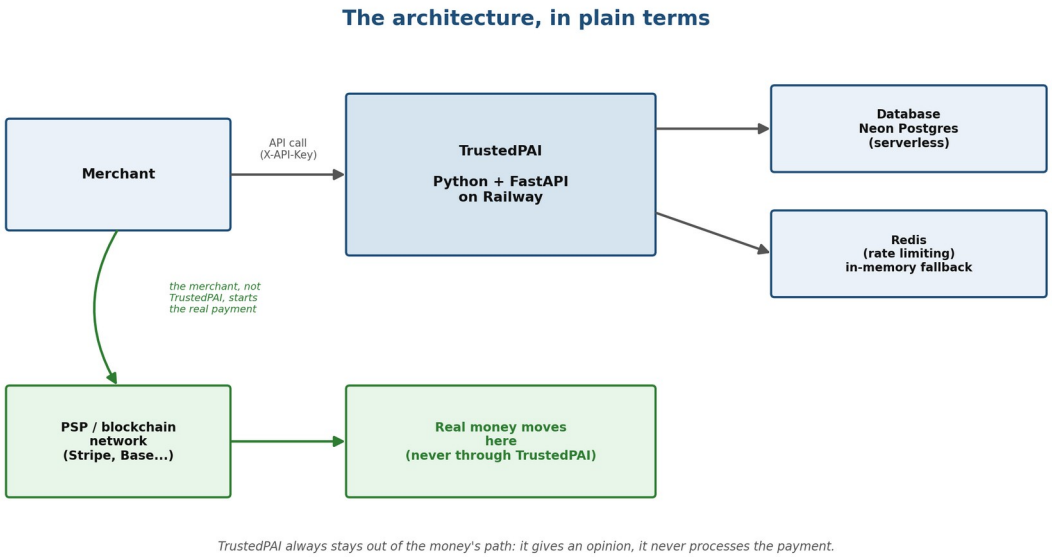
11. Applied security and technical architecture, in plain language

11.1 Security

- **Rate limiting**: each API key can make at most 60 requests per minute. The check happens BEFORE the key itself is verified, deliberately: so that not even an attempt with a wrong/made-up key can be used to “try to guess” valid keys in rapid succession.
- **Anti-replay**: for each protocol, TrustedPAI keeps track of authorization identifiers already seen (ACP tokens, AP2 PaymentMandate identifiers, sender+nonce pairs for x402, channel+cumulative-amount for MPP/tempo) to block every reuse attempt.
- **Encryption of customer keys**: the Stripe secret keys customers register are encrypted in the database (Fernet algorithm), never stored in plain text; they never travel in the body of a single screening request.
- **Fail-safe, not fail-open**: as already stated in sections 4 and 8: an internal error always produces a cautious HOLD, never an automatic approval by default.
- **Error monitoring**: an error-tracking system (Sentry) automatically flags when something breaks in the code, explicitly excluding actual payment data from what's sent to that external service.

11.2 Technical stack

A simple way to see how the technical pieces fit together:



The store talks to TrustedPAI via API; the actual money always moves through a separate path.

- **Language and framework**: Python, with FastAPI (one of the most widely used frameworks for building fast-responding web services).
- **Hosting**: Railway, a service that hosts and runs the code: no need to manage physical servers or complex infrastructure configuration.
- **Database**: Neon, a “serverless” Postgres (it scales itself to load, costs little when lightly used), currently on the free tier, with the caveat that it can temporarily suspend itself after prolonged inactivity.
- **Request queue/throttling**: Redis (a very fast in-memory database), used for rate limiting shared across the running copies of the service; if Redis is unavailable, the system automatically falls back to a simpler internal count, instead of stopping working altogether.

11.3 How it's tested

Every protocol has an automated test suite that runs against a real server and a real database (not fake simulations): it checks both “clean” cases (a legitimate transaction correctly approved) and attack scenarios (fake signature, tampered amount, reused authorization, agent with a history of abuse). For parts that require real external services that are hard to test automatically (e.g. Stripe's actual sandbox, a public blockchain node), those individual external pieces are replaced with equivalent checks in a controlled test environment; never the cryptographic verification logic itself, which is always the real thing.

12. What TrustedPAI does NOT do (to be fully clear)

- It never handles money directly: it isn't a payment processor, it never touches credit cards or wallets.
- It doesn't create or own AI agents: it isn't OpenAI, it isn't Google; it's independent of whoever builds the agent.
- It doesn't decide on the store's behalf: it gives advice (APPROVE / HOLD / REJECT), but the final decision to proceed always remains the store's or its PSP's.
- It never guarantees 100% cryptographic verification in every mode: when a protocol itself (e.g. MPP/“card”) doesn't allow it by design, TrustedPAI states this explicitly instead of pretending otherwise.
- It isn't yet a strictly “enterprise-ready” product; see section 14 on honest limitations.

13. Frequently asked questions

A collection of the most natural questions that come to mind reading this document for the first time, with direct answers.

Can TrustedPAI steal or hold on to my money?

No. TrustedPAI never touches money at any stage: it doesn't receive it, doesn't hold it, doesn't transfer it. The actual payment always happens solely through a real payment processor (Stripe, a blockchain network). TrustedPAI only gives a verdict BEFORE that payment is carried out by someone else.

What happens if the AI agent makes a mistake and buys something I didn't want?

It depends on where the error lies. If the original request was ambiguous and the agent misread it (section 2, the “good-faith error” category), TrustedPAI has no way to know: the signatures are valid, the transaction falls within policy, and the verdict will be APPROVE, because from TrustedPAI's point of view everything is in order. This is a real, honest limitation: TrustedPAI verifies that the AUTHORIZATION is genuine and falls within the risk rules, it can't read the user's mind to know what they actually meant. Protection against this kind of error lies further upstream, in how the user phrases instructions and in the spending limits set on the agent itself.

Does TrustedPAI know who I am, as a person?

TrustedPAI receives the identifiers already present in the payment mandates/tokens (e.g. an agent identifier, a reference to a tokenized payment method), not a broad profile of the person. It isn't an identity system: it relies on credentials already verified by others (the credential-issuing entity in AP2, Stripe in ACP) rather than building its own record of end users.

Who pays if TrustedPAI gets it wrong and approves a transaction that later turns out to be fraud?

Today, without a formal service-level agreement (SLA) or professional liability insurance (see section 14.1), the financial responsibility for a wrong decision stays with the store that chose to trust the verdict, exactly as, today, a traditional fraud system provides a score, but the final decision and the responsibility remain with the store that uses it. This is one of the reasons why, in section 15, the idea of insured guarantees on the risk decision is listed as a future development, not something already in place.

Why isn't it enough to use Stripe Radar or an existing traditional fraud system?

Traditional fraud systems are optimized to recognize anomalous human behavior: typing patterns, mouse movement, browsing history, device used. An AI agent doesn't generate these signals, or generates them in a completely different way from a human, so those models aren't automatically effective. TrustedPAI, by contrast, is built around signals specific to agentic commerce: the cryptographic validity of a signed mandate, the history of an agent identifier, compliance with a protocol like AP2/ACP/x402/MPP. They're complementary, not substitutes for one another.

Does TrustedPAI only work when an AI is involved?

Yes, in the sense that it's specifically designed for the case where the one authorizing the payment is an agent acting on a person's behalf, through one of the supported protocols. A payment made directly by a human on a normal site, with no agent involved, doesn't go through any of the four screening endpoints; that's not the use case it exists for.

What happens if a sixth protocol comes out tomorrow?

Technically, adding a new protocol means building a new verification endpoint specific to that signature/mandate format, reusing the same already-existing risk engine (sections 8 and 10 don't change). That's exactly the path already followed four times (AP2, ACP, x402, MPP) and, prospectively, a fifth time for UCP when/if it becomes relevant (section 5.5). The architecture was designed from the start to be protocol-neutral, for exactly this reason.

Can a store ignore TrustedPAI's verdict and proceed anyway?

Yes. TrustedPAI gives advice, it has no technical power to block a payment: it doesn't execute it. A store could, in theory, receive a REJECT and process the payment anyway through its own processor. That would be a choice by the store that defeats the purpose of the service, but it remains technically possible: TrustedPAI is an advisor, not a gatekeeper with absolute veto power over the flow of money.

Is it safe to hand Stripe secret keys over to TrustedPAI?

Keys are encrypted in the database with the Fernet algorithm (section 11.1) and are never included in the body of individual screening requests. That said, as honestly noted in section 14, there is not yet an independent security audit nor a certification (SOC 2, ISO 27001) confirming this from an external third party:

today the guarantee rests on an honest description of how the code works, not on a certified external check.

Is TrustedPAI open source?

The protocols TrustedPAI implements (AP2, ACP, x402, MPP) are open, public standards, available for anyone to consult. TrustedPAI's own specific code (the risk engine, the policies, the infrastructure) isn't made public today.

How does TrustedPAI actually make money?

The model under discussion, described in full in section 15, has whoever receives the payment (the store or its PSP) paying, not whoever owns the agent nor the underlying payment infrastructure. The concrete path toward the first real revenue (a pilot, then a usage-based model) is mapped out but not yet started, by deliberate choice (finish the product first).

What happens if TrustedPAI's database breaks right while it's evaluating a transaction?

As explained in section 4 and restated in section 8: the system is designed to respond regardless, with an explicit and justified HOLD (“automated evaluation failed, held for safety”), not with a silent error nor with a fallback automatic approval. It's a deliberate design choice, called “fail-safe”: in case of doubt or failure, the system always leans toward caution, never toward letting things slide.

What does “beta” actually mean for x402 and MPP?

It means the basic cryptographic verification works and has been tested, but coverage isn't yet as complete as AP2's and ACP's (“stable”): for x402, networks like Solana or “mainnet” Ethereum are missing; for MPP, the “tempo” mode doesn't yet read channel state directly on the blockchain (section 5.4). It doesn't mean “not working,” it means “partial coverage, explicitly stated as such.”

Who verifies that TrustedPAI itself is honest and secure?

Today, honestly, no external third party has certified it yet; this is one of the most important points in section 14. There's no independent audit yet, no industry certification. This is one of the reasons this very document deliberately includes an entire chapter on the project's real limitations, instead of only telling the best part of the story.

Does the project stop here, with what's written in this document?

No, quite the opposite: it's nearly certain several details in here will change in the coming months, given how fast the entire agentic-commerce industry is moving (section 3). This document captures a snapshot of the project and the protocols as they stand in mid-2026: it's a good starting point for understanding the underlying logic, not a text that will remain accurate in every detail forever.

14. Honest state of the project: what exists today and what's still missing

This section exists specifically to avoid only telling the best part of the story. It's organized by area.

14.1 Legal and corporate

- No legal entity has been set up (today it's a solo individual's project, not a company).
- No formal ToS (terms of service) or SLA (service-level agreement).
- No DPA (data processing agreement, relevant under European GDPR).
- No professional liability insurance.

14.2 Technical trust

- No independent security audit has been carried out yet.
- No certification (e.g. SOC 2, ISO 27001) obtained.
- Database on Neon's free tier, which can suspend itself after prolonged inactivity.
- Encryption of transactional data “at rest” (beyond the Stripe keys, already encrypted) was considered and deliberately deferred: the underlying database is already encrypted at the platform level by Neon, and encrypting at the application level too would break the dashboard's text search without a clear security benefit (a deliberate choice, not an oversight).

14.3 Operational reliability

- No real-time monitoring of risk anomalies or service downtime (uptime): only application error tracking.
- No automatic alerts (e.g. to Slack or email) when something goes wrong.
- No stated availability SLA.
- A single point of failure on the database; no disaster-recovery system tested yet.

14.4 Business

- No real pilot customer yet, a deliberate decision: finish the features above first, then look for a first customer.
- No automated usage-based billing.
- No dedicated SDK for people building AI agents (e.g. for frameworks like LangChain).
- No official website yet: the domain trustedpai.com is owned, but a real site will only be built once the product matures, ready to be presented and sold.

14.5 Technical coverage

- 4 of the 5 relevant payment protocols covered; only UCP is missing (see section 5.5).
- AP2 supports only one trusted “issuer” (credential-issuing entity) per customer; a true multi-issuer trust registry doesn't exist yet.
- x402 limited to Ethereum-compatible blockchain networks (no Solana, no “mainnet” Ethereum).
- MPP/“tempo” doesn't yet read the payment channel's state directly on the blockchain.
- Onboarding a new customer is still manual (done by hand); only the modification of risk rules, once the account exists, is already self-service.

14.6 What it would take to sell to large/enterprise companies

A more extensive list, aimed specifically at the most demanding kind of customer (banks, large retailers, regulated companies), deliberately not pursued until a concrete request comes from a real customer, so as not to invest time in things nobody has asked for yet:

- SOC 2 Type II certification, periodic penetration testing, a public vulnerability disclosure policy.
- A register of sub-processors (who handles data on TrustedPAI's behalf), audit logs, a formal incident-response and data-breach notification plan.

- For regulated financial customers in the European Union: DORA compliance (mandatory since 2025, often required contractually).
- Standard MSA/SLA contracts, E&O/cyber liability insurance, a registered trademark.
- A real uptime SLA with contractual penalties, a public status page, elimination of the single point of failure on the database, a genuinely tested disaster-recovery plan, an on-call process.
- A separate sandbox environment for enterprise customers, asynchronous webhooks, SSO/SAML support for corporate login.
- Pre-filled industry-standard security questionnaires (CAIQ/SIG), public case studies from real customers.

14.7 Interesting ideas, deliberately deferred

- Real-time confirmation to the end user, above a certain risk or amount threshold.
- Natural-language explanations of decisions (also relevant to European AI regulation, the AI Act).
- A true shared reputation network across different customers, with its own dedicated logic; today history is shared “in practice” for the same agent, but without a network/product of its own.
- Correlating anomalies ACROSS different agents/stores (not just a single agent's own history), to uncover coordinated fraud patterns: an idea also backed by independent research into AP2's security, read during development.
- An insured guarantee on the risk decision.
- A dedicated SDK/plugin for the major AI-agent-building frameworks.

15. The goal behind the project

TrustedPAI isn't meant to simply generate a small, steady monthly income. The stated goal is to build a solid, “attractive” company, one that could eventually be acquired by a larger player in the payments industry: a bank, a card network, a payment-risk-management firm.

For this reason everything (the code, the technical documentation, the field names in the API, even the project's public identity: username, commit author) has been written and organized from the outset with an international audience and market in mind, not just an Italian one: a stray Italian detail here or there would have been a real obstacle during a future due-diligence process by a non-Italian acquirer.

The go-to-market path under discussion, though not yet started, involves: first a “shadow” pilot with 1-2 customers found among those already working with ACP/AP2; then a usage-based API pricing model; then possible integrations with PSPs or agencies that already manage Shopify stores as a distribution channel; finally, further down the line, possible insurance guarantees built on top of the risk history accumulated over time. It's deliberate that none of these steps has been started yet: the decision made was to finish the “arsenal” features (section 10) and protocol coverage (section 5) first, so as not to show up in front of a first customer with a half-built product.

Anyone reaching the end of this document should now have everything they need to understand not just WHAT TrustedPAI does today, but also WHY it's built this way, where its real boundaries are, and in what direction it's meant to grow.

Appendix: Complete glossary, in alphabetical order

All the technical terms used in the document, gathered here for quick reference.

- **AI agent**: an artificial-intelligence-based program capable of carrying out autonomous actions (browsing sites, filling out forms, completing purchases) to reach a goal set by a person, without needing step-by-step confirmation.
- **API**: “Application Programming Interface”: a standard way for two programs to talk to each other over the internet, exchanging messages in an agreed-upon format (usually JSON).
- **Blockchain**: a shared, public ledger of transactions, jointly maintained by many different computers, where every recorded operation is practically impossible to erase or alter afterward.
- **Chargeback**: a request, made by a cardholder to their own bank, to reverse a payment already made and get the money back, typically over a dispute or fraud.
- **Public key / private key**: a pair of mathematically linked codes: the private key stays secret and is used to sign, the public key can be shared with anyone and is used to verify that a signature is genuine.
- **Agentic commerce**: the slice of online commerce where the purchase is initiated or completed by an AI agent on a person's behalf, instead of by a direct human click at that precise moment.
- **Verifiable digital credential**: a cryptographically signed digital document attesting to a fact (e.g. “this payment is authorized by this user”) that anyone can verify mathematically without needing to contact whoever issued it.
- **ECDSA**: a cryptographic signature algorithm based on elliptic-curve mathematics, underlying many modern digital signatures, including those used by blockchain wallets.
- **EIP-712**: an Ethereum standard for signing structured, human-readable messages (not just raw blocks of data), used by both x402 and MPP/“tempo.”
- **Endpoint**: a specific API address a program can send a request to in order to get a certain service.

- **Fail-safe**: a design principle whereby, in case of error or doubt, the system always chooses the more cautious option (e.g. blocking or holding), never the more permissive one.
- **Cryptographic signature / digital signature**: a mathematical code generated with a secret key, attached to a message, that proves who created that message and that no one has modified it since.
- **Hash (digital fingerprint)**: a fixed-length code calculated from any given content: even the smallest change to the original content produces a completely different hash, which makes it useful for detecting tampering.
- **HTTP**: the standard protocol browsers and applications use to communicate with web servers, underlying the internet as we know it.
- **JWT (JSON Web Token)**: a widely used standard format for representing digitally signed information compactly, used for example by AP2's CartMandate.
- **JWE (JSON Web Encryption)**: a standard for encrypting (not just signing) a message, so that only whoever holds the right key can read its contents, used by MPP's “card” mode.
- **L2 / Layer 2**: a blockchain network built “on top of” a main network (typically Ethereum) to be much faster and cheaper, while still keeping a strong security link to the main network.
- **Mandate**: in AP2's terminology, a cryptographically signed digital contract that authorizes a specific part of the purchase process (intent, cart, payment).
- **MCP (Model Context Protocol)**: an open standard created by Anthropic that lets an AI assistant use external tools in a structured way, as it would with a shared toolbox.
- **Merchant**: the store, i.e. whoever sells the product or service and receives the payment.
- **Nonce**: a number (or code) used only once within a cryptographic protocol, to guarantee that every message/authorization is unique and can't be reused.

- **OAuth 2.0**: a widely used standard for delegating access permissions securely and revocably, without directly sharing one's own credentials (e.g. “sign in with Google”).
- **Onboarding**: the process by which a new customer (a store) is registered and configured on a service for the first time.
- **PSP (Payment Service Provider)**: the company that technically handles payments on a store's behalf (e.g. Stripe): collects funds, manages cards, handles the actual financial side.
- **Rate limiting**: a mechanism that limits how many requests a single user/key can make within a given time window, to prevent abuse and automated attacks.
- **Replay attack**: an attack in which someone intercepts a payment authorization already legitimately used once, and tries to resend it identically a second time to pay twice with the same signature.
- **Sandbox**: a test environment separate from the real one, where an integration or a payment flow can be tested without moving real money.
- **SD-JWT-VC**: a verifiable digital credential format, based on JWT, designed to selectively disclose only some of the information it contains; used by AP2's PaymentMandate.
- **Stablecoin**: a cryptocurrency whose value is pegged to a traditional currency (usually the US dollar), designed not to fluctuate the way Bitcoin does: 1 USDC stablecoin is always worth roughly 1 dollar.
- **Token**: a code that securely represents a right or permission (e.g. the right to make a specific payment), without directly exposing the sensitive data underneath.
- **TTL (Time To Live)**: the maximum length of time an authorization or piece of data stays valid, after which it automatically stops working.
- **Wallet**: a “digital wallet”: an address on a blockchain, controlled by a private key, where cryptocurrency funds are held and moved.
- **Webhook**: a mechanism by which one service automatically notifies another service when a certain event happens (e.g. “order shipped”), without the latter having to keep asking for updates.

- **X-API-Key**: the name of the HTTP header TrustedPAI uses to receive the authentication key assigned to each customer.

Index