RelAI Whitepaper

HTTP-native micropayments and on-chain privacy for the agentic web

Version 1.0 | June 2026 | relai.fi

Abstract

RelAI is a multi-chain marketplace that turns any HTTP API into a metered, pay-per-call service using the **x402** payment standard. A caller (a human app or an autonomous AI agent) requests a resource, receives an HTTP `402 Payment Required` challenge, signs a payment, and retries. RelAI verifies the payment, proxies the upstream request, and settles on chain in a single round trip. Gas is sponsored by RelAI, so callers sign but never need native tokens, and API providers never run payment infrastructure of their own.

On top of this payment rail, RelAI adds a **zero-knowledge privacy layer**: shielded pools, private payment requests, and bearer payment codes let value move with the amounts and identities hidden on chain, while an **Association Set Provider (ASP)** keeps the system compliant by cryptographically excluding screened-out deposits rather than freezing funds. The result is an instant-settlement payment fabric for APIs and AI agents that is multi-chain, gas-free for users, optionally private, and compliant by construction.

This document describes the protocol, the architecture, the privacy and cryptography design, the compliance model, the agent-identity layer, the deployed contracts and their security posture, the economics, and the roadmap.

---

1. Introduction

1.1 The problem

The web has never had a native way to charge for a single request. HTTP reserved status code `402 Payment Required` in 1997 and then left it unused for a quarter century. As a result, API monetization grew up around API keys, monthly subscriptions, prepaid credits, and invoicing: all of which assume a human signs up in advance, shares a card, and reconciles bills later.

Two shifts make this model obsolete:

1. **APIs are the product.** Data feeds, inference endpoints, search, storage, and tools are sold by the call. Subscriptions misprice them: light users overpay, heavy users get rate-limited, and long-tail providers cannot monetize at all because onboarding friction dwarfs a fractional-cent call.

2. **AI agents are the new caller.** Autonomous agents need to discover, pay for, and consume services without a human in the loop. An agent cannot fill in a credit-card form, cannot wait for a monthly invoice, and should not be handed a long-lived secret with unbounded spend.

What is missing is a machine-native, per-call settlement primitive: instant, programmable, low-fixed-cost, and usable by software that holds a key but not a bank account.

1.2 The privacy gap

Putting payments on chain solves settlement but creates a new problem: every payment is public. The amount, the payer, the payee, and the timing of each API call become a permanent, queryable record. For salary-like payouts, competitive usage patterns, or simply user dignity, fully transparent payment graphs are a regression from the web2 status quo, not an improvement.

Existing privacy tools force a bad trade. Fixed-denomination mixers break composability and invite blanket sanctions. Fully opaque systems cannot satisfy compliance and get delisted. RelAI's thesis is that privacy and compliance are not opposites: with the right cryptography, honest users can transact privately while non-compliant funds are excluded by math, not by a custodian's freeze switch.

1.3 What RelAI is

RelAI is three things in one stack:

- **An x402 API marketplace and relay.** Providers list APIs; callers pay per request through a shared gateway that speaks both x402 v1 and v2 across eleven chains.
- **A gas-sponsored, multi-chain settlement layer.** Users sign payment authorizations; RelAI's backend pays the gas and settles, using EIP-3009 on EVM chains and fee-payer sponsorship on Solana.
- **An optional privacy and compliance layer.** Shielded pools, private payment requests, and payment codes hide amounts and identities on chain, with an Association Set Provider enforcing sanctions compliance cryptographically.

Everything is unified by one idea: payment should be a property of a request, not a relationship.

---

2. Design principles

- **HTTP-native.** Payment is expressed in the request/response cycle (`402` plus a payment header), so any client that speaks HTTP can pay, and any API can be wrapped without an SDK rewrite.
- **No chain lock-in.** v1 and v2 of x402 coexist, and RelAI settles across every supported chain, so providers and callers are not bound to a single network or custody model.
- **Users sign, RelAI pays.** Callers never hold native gas tokens. They sign an authorization off chain; the backend sponsors and submits the transaction.
- **Privacy is opt-in and cryptographic.** Confidentiality comes from zero-knowledge proofs and Merkle commitments, not from trusting RelAI to keep a secret database.

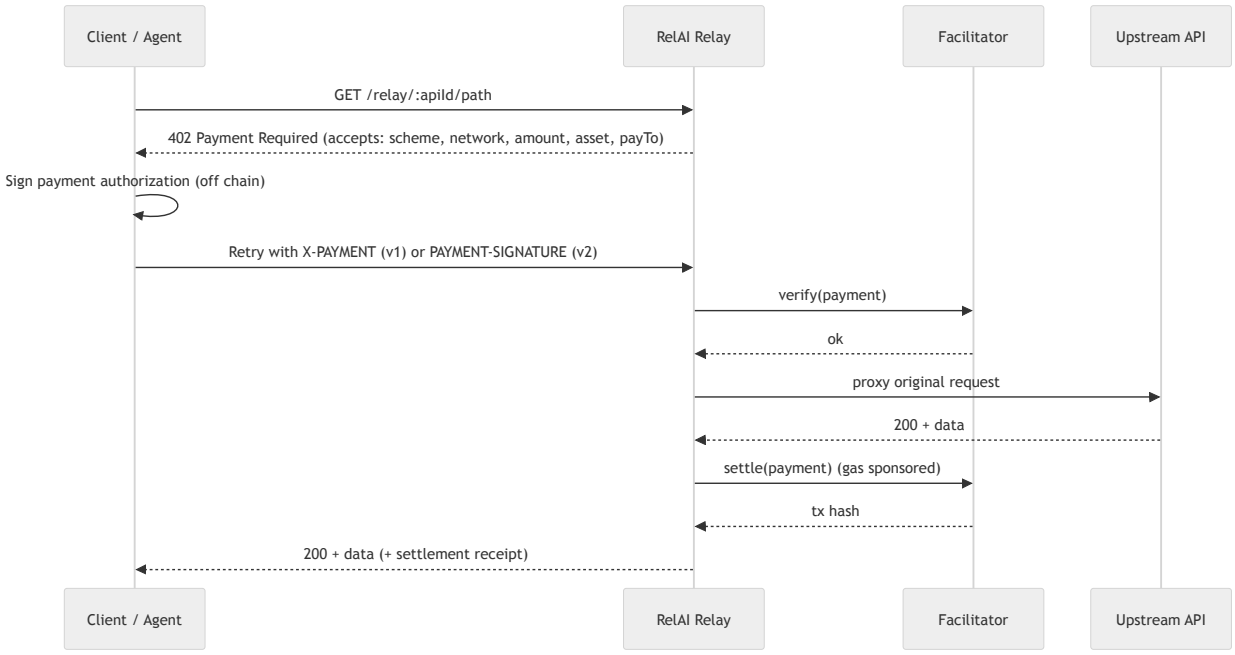
- **Compliance without custody.** RelAI cannot freeze or seize funds. Compliance is a membership constraint inside the proof: screened-out deposits simply cannot produce a valid spend.
- **Defense in depth.** Timelocks on privileged config, code-pinned verifiers, rate limits, replay guards, and independent audits harden every layer.

---

3. The x402 protocol

3.1 Challenge and response

The core flow is a single retry loop over standard HTTP:



The `402` body carries an `accepts` array describing one or more ways to pay: the `scheme`, the `network` (as a CAIP-2 identifier), the `amount` in atomic units, the `asset` (token), the `payTo` address, and a timeout. The client picks one, produces a proof, and retries.

3.2 Two protocol versions

- **x402 v1** uses an `X-PAYMENT` header and a URL-based facilitator. It is the simplest integration path and is handled by the `x402-express` / `x402-axios` family of SDKs.
- **x402 v2** (the Coinbase-aligned spec) uses a structured challenge and a `PAYMENT-SIGNATURE` header that can carry a signature chain. It supports discovery and quote extensions and is handled by the x402 v2 SDK stack (`@x402/express`, `@x402/svm`).

Both versions run side by side; an API selects its version, network, and facilitator at the entity level.

3.3 Networks and schemes

RelAI settles across eleven chains today: **Solana** (mainnet and devnet), **Base**, **Base Sepolia**, **Polygon**, **Ethereum**, **SKALE Base**, **Avalanche**, **Tempo**, **Telos**, **SEI**, and **peaq**. Chains are identified with CAIP-2 (for example `eip155:8453` for Base, `solana:5eykt4UsFv8P8NJdTREpY1vzqKqZKvdp` for Solana mainnet).

Payment schemes are scheme-tagged: `exact` for EVM v1, `evm-exact` for EVM v2, and `svm-exact` for Solana v2. USDC is the default settlement asset (for example `EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v` on Solana, `0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913` on Base), with other SPL and ERC-20 assets configurable per API.

3.4 The RelAI facilitator

A facilitator is the service that verifies a payment and settles it on chain. RelAI runs its own first-party facilitator at `facilitator.x402.fi`, and it carries all traffic: every payment is verified and settled through RelAI across all supported chains (Base, SKALE Base, Avalanche, Solana, Polygon, Ethereum, Telos, and others). Running settlement in-house is what makes gas sponsorship native (Section 3.5): RelAI is the account that pays the network fee, so the caller only ever signs.

The relay keeps a facilitator-agnostic interface (the payment-plan resolver abstracts verification and settlement behind a common shape, so the surface is not tied to one backend), but in production RelAI's own facilitator is the single settlement path. An API that already speaks x402 upstream can still be proxied as-is.

3.5 Settlement and gas sponsorship

The defining user-experience property of RelAI is that **callers never pay gas**:

- **EVM.** The client signs an EIP-3009 `TransferWithAuthorization` permit off chain (no gas, no prior approval). RelAI's backend wallet executes `transferWithAuthorization` on chain and pays the native fee (ETH on Base/Ethereum, AVAX on Avalanche, CREDIT on SKALE, and so on).
- **Solana.** The client signs a transaction in which RelAI's backend account is the `feePayer`. RelAI submits it, sponsoring the lamports.

Pricing is resolved per request from a USD figure stored on the API. The Solana pricing engine converts USD into SPL base units using live quotes; the EVM engine passes a dollar-denominated amount to the facilitator. Quotes carry a 60-second TTL and are re-issued on retry. The platform-wide minimum is **$0.01** per call.

3.6 WebSocket relay

For agents and long-lived clients, a shared WebSocket gateway at `wss://api.relai.fi/api/ws/relay` mirrors the HTTP flow over JSON-RPC: a `relay.call` returns a `402`-equivalent `paymentRequired` error, the client retries with a `payment` field, and the gateway forwards to the same relay path internally. Providers do not run their own socket servers.

3.7 Free tier

Each API can expose a free tier with a per-caller allowance and a global cap, resetting never, daily, or monthly. The `402` response carries the remaining counts as an x402 extension, so a caller (or an agent) can discover how many free calls remain before payment is required. This is the on-ramp that lets agents try a service before committing funds.

3.8 Recurring payments: subscriptions

Per-call settlement is the default, but many services bill on a recurring basis. RelAI adds native on-chain subscriptions on Solana through a dedicated subscription program, built so that a subscriber signs exactly once.

A merchant creates a plan that fixes a per-period amount, a token, and a period length; RelAI is the plan owner and the authorized puller, and the merchant wallet is the payout destination. A subscriber joins with a single transaction (initialize a subscription authority, then subscribe). After that no further signature is needed: RelAI's billing scheduler (the "puller") periodically charges every due subscription.

The authorization is bounded and policed on chain, not a blanket allowance. The program tracks the current period and the amount already pulled within it, permits the first charge immediately and each later charge only once the period has elapsed, and refuses to pull more than the plan amount per period. The subscriber can cancel or resume at any time. Every charge settles in a single transaction that splits the funds natively: the merchant receives the net and the RelAI platform fee (5 percent) routes to the fee wallet. Plans run on Solana mainnet, with a devnet tier for testing.

The effect is recurring revenue for providers with the trust profile of self-custody: the subscriber grants a capped, revocable, per-period pull that the program enforces, rather than handing any party an open-ended spending approval.

---

4. Marketplace and developer experience

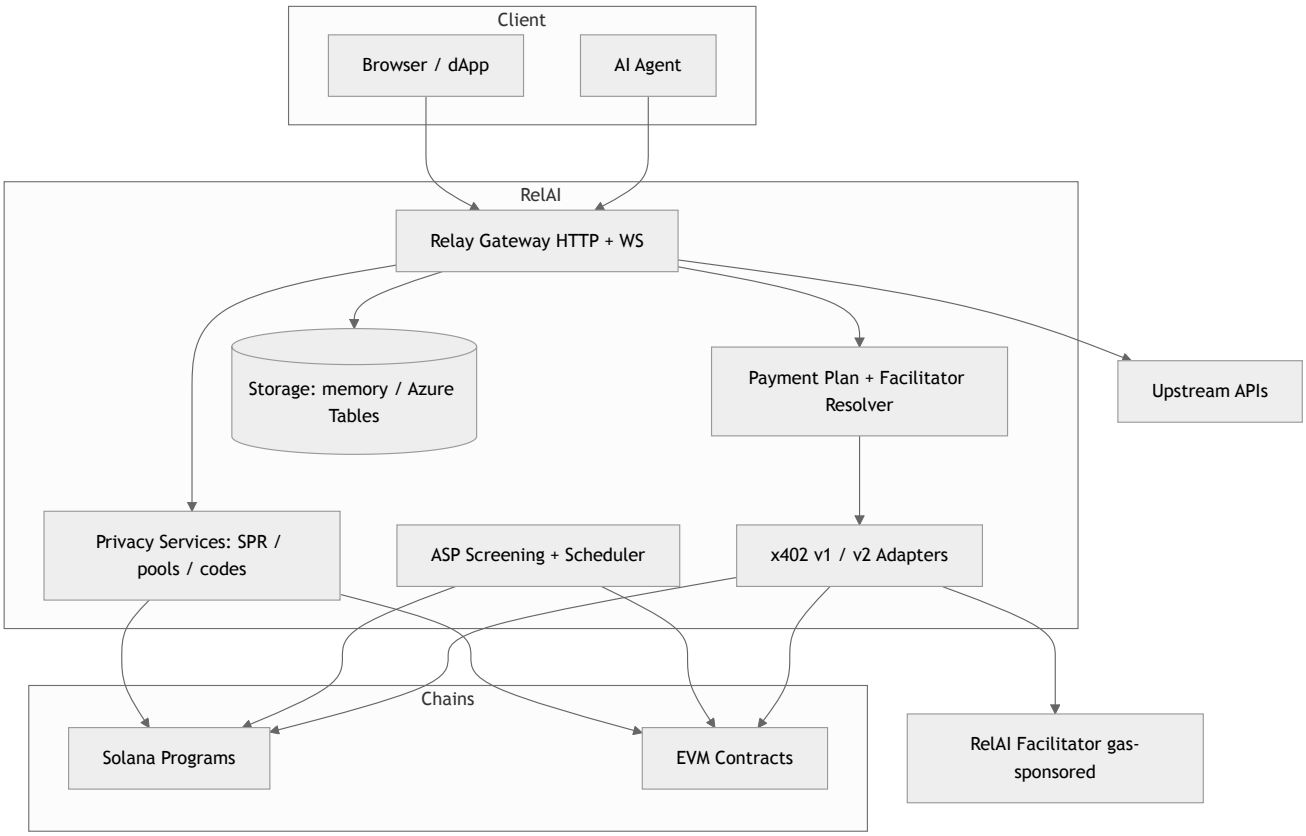
RelAI is not only a protocol; it is a product surface that removes operational work from both sides of a transaction.

- **Relay gateway.** Providers register an API (its base URL, network, pricing, and OpenAPI metadata). RelAI exposes `GET|POST /relay/:apiId/*`, handles the `402` handshake, verifies and settles, and proxies the upstream call. Providers ship API logic, not payment servers.
- **Paywall.** A generic, embeddable paywall client accepts an API id, resource, amount, and branding as query parameters and drives the full pay-and-fetch loop. In embed mode it emits `postMessage` success and close events, so any website can monetize a resource with an iframe.
- **Marketplace, playground, and bazaar.** A browsable catalog (`/market`), an interactive tester (`/playground`), and an agent-facing discovery surface (`/bazaar`) make APIs findable by humans and machines.

- **Earnings and settlement.** Provider earnings accrue to a virtual balance keyed by owner, network, and atomic amount. Withdrawals are batched (one on-chain payout per accumulated threshold), keeping gas overhead well under one percent even for fractional-cent calls. An optional Stripe bridge can route a fiat share to providers who prefer cards.
- **Webhooks and provenance.** On settlement, RelAI can fire a provider webhook with the transaction, network, amount, payer, recipient, and endpoint, and can stamp a provenance record (see Section 9) so a provider can prove a call was paid without revealing the caller.

---

5. System architecture



- **Frontend.** Next.js 15 / React 19 / TypeScript at `relai.fi` (production) and `int.relai.fi` (staging). Multi-wallet by design: Solana Wallet Adapter, Reown AppKit, Wagmi, and Crossmint embedded wallets, all SSR-safe. ZK proving runs client-side with `snarkjs` and `circomlibjs`.
- **Server.** Node 20 / Express (ESM) with roughly sixty route modules mounted behind a dynamic relay. Storage swaps between an in-memory store and Azure Tables. A shared WebSocket relay, an ASP scheduler, and provenance initialization run alongside the HTTP server.
- **Contracts.** Solidity contracts on the EVM chains and Anchor programs on Solana, plus Circom circuits compiled to Groth16 verifiers for both ecosystems.
- **Custody options.** For recipients who are not yet on chain (for example the bearer of a payment code), RelAI can manage an encrypted keypair on the user's behalf, exportable at any time for self-custody.

---

6. The privacy system

RelAI's privacy layer is built from one primitive (a shielded pool) and three products that use it: shielded payment requests, the private gift card, and bearer payment codes.

6.1 Goals and threat model

The privacy layer is designed so that the parties RelAI itself runs are **not trusted with privacy secrets**. The relayer that sponsors gas, the indexer that rebuilds trees, and the ASP operator that screens deposits all see only public commitments and nullifiers. Only the prover's own device (browser or mobile) holds the secrets that could deanonymize a transaction. The threats addressed include the on-chain observer (amounts and identities), the storage breach (data at rest), and network-level correlation (IP linkage), each targeted by a specific mitigation in the roadmap (Section 12).

6.2 Shielded pools

A shielded pool is a UTXO system over a Poseidon Merkle tree. Depositing inserts a **commitment** (a hash of secret note material) as a leaf; spending proves, in zero knowledge, knowledge of a leaf and reveals a **nullifier** that marks it spent without revealing which leaf it was. Amounts live inside the commitment, never in the clear.

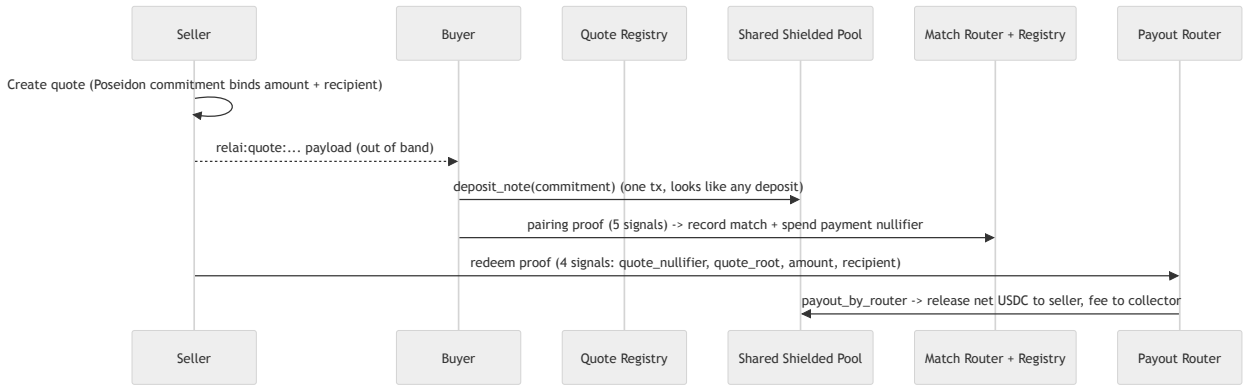
RelAI runs two production implementations sharing the same commitment model:

- **EVM: ShieldedPool V4** (with V2/V3 predecessors), an incremental Poseidon-Merkle pool with ASP-gated withdrawals, an issuer fee at deposit, and a relayer fee at withdraw.
- **Solana: Private Pool V2**, a Sapling-style system with a 32-deep Merkle tree (about four billion leaves of capacity), three Groth16 circuits (Deposit, Withdraw, and a 2-in-2-out JoinSplit), encrypted on-chain note blobs, and compliance-aware nullifier spending. A second Solana program, `solana-shielded-pool` (a 20-deep tree), backs the shielded-link and SPR flows.

The note commitment is a width-7 Poseidon hash, `Poseidon(noteVersion, poolIdHash, assetIdHash, amount, secret, blinding, nonce)`, identical across products so that different flows share one anonymity set.

6.3 Shielded Payment Requests (SPR)

SPR is private direct payment: a buyer pays a seller with no on-chain amount or identity, and no separate deposit-then-withdraw round trip that a watcher could link.



The mechanism, end to end:

1. **Quote.** The seller's backend creates a quote whose secret binds the amount and the seller's identity: a Poseidon commitment `Poseidon(amount, sellerSecret, nonce, quoteId)` and a deterministic quote nullifier `Poseidon(sellerSecret, nonce, quoteId)`. Issued quotes are inserted into an off-chain quote Merkle tree (depth 20) whose root is published on chain to a **Quote Registry**. The quote is shared with the buyer as a compact `relai:quote:` payload, out of band.
2. **Deposit.** The buyer derives a pool commitment and calls `deposit_note` once. This is a single instruction that is indistinguishable from any other pool deposit, including shielded-link deposits, so the buyer joins a **shared anonymity set**.
3. **Pairing.** The buyer produces a Groth16 pairing proof exposing five public signals (pool root, ASP root, quote root, payment nullifier, quote nullifier). The Match Router verifies the proof, records the match in a **Payment Match Registry** (one PDA per quote, preventing re-matching), and atomically marks the payment nullifier spent in the pool.
4. **Redeem.** The seller produces a Groth16 redeem proof exposing four public signals (quote nullifier, quote root, amount, recipient). The **Payout Router** checks that a match record and a published quote root both exist (the anti-drain guarantee), verifies the proof, and releases the net amount to the seller while routing a platform fee to a collector account.

A privacy relayer signs the on-chain pairing and redeem transactions, so a block explorer shows the relayer as signer rather than the buyer or seller. To a chain observer, "Alice paid Bob five dollars" is indistinguishable from a self-split or a shielded-link exit.

6.4 Payment codes (the private gift card)

Payment codes are bearer instruments: a short, human-typable code (BLIK-style, eight characters from a keyspace of roughly 1.1 trillion) that anyone can redeem to any wallet, with the amount hidden on chain. The payer pre-funds an on-chain escrow; the redeemer needs no prior wallet and no gas.

A security-critical design choice (hardened in audit) is that the on-chain vault is keyed not by the short, brute-forceable code but by a 256-bit high-entropy `vault_id` used as the PDA/escrow seed. The short code stays off chain; the server gates redemption and rate-limits attempts (five per minute, per IP and per code), closing offline brute-force. The same `vault_id`, being public, enables permissionless cross-device cancel so a buyer can reclaim unspent funds from any device. Codes settle on EVM via an

`EvmPaymentCodeEscrow` (funds locked until redeemed) and on Solana via an Anchor escrow, with a one percent fee on Solana-to-EVM cross-chain redemption.

---

7. Zero-knowledge cryptography

7.1 Building blocks

RelAI's privacy proofs are **Groth16** over the **BN254** curve (scalar field prime `21888242871839275222246405745257275088548364400416034343698204186575808495617`), authored in **Circom 2.1.6**. Hashing is **Poseidon** at several widths: width 2 for Merkle nodes, width 3 for the quote nullifier, width 4 for the quote commitment and amount binding, and width 7 for pool note commitments. Each proof serializes to a compact 256 bytes (64 + 128 + 64 for the A, B, C group elements).

7.2 Circuits

Circuit	Public signals	Purpose
ShieldedWithdraw	6	Pool membership + nullifier for a shielded withdrawal
ShieldedWithdrawWithAsp	7	As above plus an ASP-root membership constraint
ShieldedPaymentPairing	5	Proves pool, ASP, and quote membership and binds the payment to a quote
ShieldedPaymentRedeem	4	Binds the claimed amount and recipient to the quote commitment
ShieldedPaymentReceipt	2	Selective disclosure of a quote commitment for a given amount

The pairing and redeem circuits are the heart of SPR. The pairing proof shows that a buyer's deposit, an ASP approval, and a seller's quote all sit in their respective published trees, and binds the payment and quote nullifiers. The redeem proof, hardened in audit finding F01, cryptographically binds the claimed `amount` and `recipient` into the quote commitment, so a seller cannot over-claim against the shared vault without faking both the amount witness and a historical root.

7.3 Verifiers

The same proof semantics run on both ecosystems. On EVM, snarkjs-generated Solidity verifiers check proofs natively. On Solana, the `groth16-solana` library performs pairing checks; verifying keys are embedded at compile time (read from `verification_key.json` by `build.rs` and baked into the program as static data, with no runtime loading). Public inputs are encoded canonically (a `u64` amount becomes a 32-byte big-endian field element; a Solana public key is reduced modulo the field), and the proof's A element is negated during decode to match the verifier's expected form.

7.4 Trusted setup

Circuits use the public Powers-of-Tau outputs (`hez_final_14` for the smaller circuits, `hez_final_16` for the Private Pool V2 family). The current devnet artifacts come from a single-contributor testnet ceremony; circuits are open source and verifying keys are committed for reproducibility. A multi-party ceremony is a prerequisite for mainnet (Section 12).

---

8. Compliance: the Association Set Provider

8.1 The Privacy Pools model

RelAI reconciles privacy with sanctions compliance using a Privacy-Pools-style **Association Set Provider**. Alongside the main pool tree, the ASP maintains a second Merkle tree containing only **screened, approved** deposits. A withdrawal or spend proof must demonstrate membership in **both** trees. The crucial property is that RelAI holds no freeze or seizure power: a non-compliant deposit is not confiscated, it simply cannot produce a valid spend proof because it is absent from the ASP tree. Honest users get full anonymity at zero cost; tainted funds are excluded by math.

8.2 Screening pipeline

A server-side scheduler periodically screens deposits against layered sources: the U.S. Treasury OFAC SDN list (daily fetch, cached), UN and EU consolidated lists (parsed feeds), and a Chainalysis sanctions oracle (staged). Each deposit gets a verdict of **ACCEPT**, **DEFER** (manual review), or **REJECT** (permanently excluded), with a coverage field that records which providers were consulted so a provider outage never silently flips a verdict. Only ACCEPT verdicts enter the next snapshot root. The pipeline is defense-in-depth: a missing source defers rather than approves, and an appeals process with audit logging allows remediation (a re-screened commitment becomes spendable on the next tree rebuild). No KYC data is retained, and contact details from appeals are deleted ninety days after resolution.

8.3 On-chain enforcement

The ASP tree (depth 20, about one million commitments, matching the pool) is published as a root to an on-chain **ASP Registry**. Pools enforce **root freshness**: a withdrawal that references a stale ASP root (older than the grace period, typically seven days) or a zero root is rejected. On Solana, the shielded pool keeps a ring buffer of the last 64 ASP roots so that a proof generated against a recent root stays valid even as the tree advances between pairing and redeem. Roots are republished on a cadence (a 60-minute default rebuild), and the freshness window forces a visible, auditable publication rhythm.

---

9. AI agents, identity, and provenance

RelAI's core thesis is **payments for AI agents**: software that can discover a service, pay for one call, and move on, while building a verifiable reputation. Three pieces make agents first-class:

- **Identity and reputation.** On EVM (SKALE Base), agents register against ERC-8004 Identity and Reputation registries; on Solana, the SATI provenance SDK plays the same role. After each paid call, the settlement path submits feedback signals (a binary reachability flag, a response-time measurement, and a success-rate percentage) to the on-chain reputation registry, with the endpoint included in the event. Market participants can then verify an agent's or an API's reliability from chain data rather than self-reported metrics.
- **Budget management.** `AgentBudgetVault` is an HTLC-style escrow: an owner deposits a set of hash-locked commitments and an agent redeems individual secrets to spend, without a wallet prompt per action and without exposing a long-lived key. Vaults can be perpetual or time-limited, and the owner can cancel only after expiry. `RelAISpender` complements this with EIP-2612 permit execution for delegated, bounded EVM transfers.
- **Provenance.** Provenance stamping (the Integritas extension and SATI on Solana) lets a provider prove that a call was paid for, on chain, without revealing the caller, which is exactly the property an agent marketplace needs to build trust without surveillance.

Reference agents (`shielded-agent` and `spr-agent`) demonstrate the full loop: an agent redeems a private link or routes an SPR payment and consumes an API, end to end, with no human interaction.

---

10. Smart contracts and security

10.1 Contract inventory

RelAI's on-chain footprint spans roughly twenty Anchor programs on Solana and a family of Solidity contracts across Base and SKALE. Selected production addresses:

Component	Network	Address
EvmPaymentCodeEscrow	Base mainnet	`0x3cD0FD49ddf7F15d9A092CA390893b5D0da726a7`
EvmPaymentCodeEscrow	SKALE Base mainnet	`0x4eB510525dE9855Ab0D1e5050d89beAFc5EE4Aa4`
ShieldedPool V4	Base Sepolia	`0x753Ae2Cd116e057AB4C05D587992DF7593f7F857`
QuoteRegistry	Base Sepolia	`0x6869De09ba3379EBe9caBD6d618F776aFe91a3eC`
PaymentMatchRegistry	Base Sepolia	`0x27BfC4b5e03aa2E562b011b9fA21A98Fc29B0F5d`
solana-shielded-pool	Solana devnet	`GW43ARYCQgzVmnX7Nx9mx1s8AjJSdrpAbthaMpKJU8aj`
solana-private-pool-v2	Solana devnet	`8QvQ1czEdPMTozNYh8Hxzp35mpxdA8S85kqYyNoK7twj`
spr-payout-router	Solana devnet	`Efqnuy6ytmd9X2LJBqdZBKoSSdnbQnsJXdGiGCHiUmkL`
payment-match-router-v2	Solana devnet	`A3GkAwYyfF7nwP4qa3EDkykL3wd7KB8oBNcwDbZnWoRP`
solana-escrow (payment codes)	Solana	`5DKRJkDtzpoYJjHdFC7QiatKAUqkjAHt2wurjoMJKmcJ`

(Full per-network addresses, including the EVM privacy-pool family and the Solana verifier programs, are tracked in the repository's `deployment-*.json` and `rollout-*.json` manifests.)

10.2 Security posture

RelAI subjects its on-chain code to internal autonomous security audits and remediates findings before mainnet. Two audit efforts are reflected in the current code:

- A **15-finding audit** (3 Critical, 4 High, 5 Medium, 2 Low, 1 Informational) of the SPR, shielded-pool, and payment-code stack. Its fixes are live: the redeem proof now binds amount and quote-tree membership end to end (no over-claim against the shared vault); the pairing verifier program is code-pinned so no substitute verifier can be injected; the legacy-pool wipe path refuses to run against a live, funded pool; the payout's recipient token account is bound to the proven recipient (no mid-broadcast swap); and the ASP root is checked against an on-chain allowlist before any verifier call.
- A **Private Pool V2 audit** that hardened the privacy pool and its compliance pipeline: the ASP publish endpoint was locked to cleared verdicts behind rate limits; the relay transaction allowlist was tightened to an exact program whitelist; privileged pool configuration was moved behind a timelocked propose/execute/cancel flow (roughly a 24-hour delay) instead of a single signature; event-recording endpoints now require a wallet-owned signature; and fee-payer and key-derivation paths were unified to remove drift.

Defense-in-depth is structural, not incidental: privileged configuration is timelocked, verifiers are code-pinned, every relay and publish endpoint is rate-limited, nullifier markers prevent replay, and compliance is enforced inside the proof. The remaining hardening before mainnet is operational rather than a code defect, and is tracked in the roadmap: a multi-party trusted-setup ceremony, verifier upgrade-authority immutability, and a 2-of-3 multisig for ASP authority.

---

11. Economics

11.1 Fee model

RelAI's economics are built around per-call settlement with a **$0.01** minimum (an optional provenance stamp adds a small flat fee). Provider earnings accrue to a virtual balance and are withdrawn in batches, so the on-chain gas cost is amortized to well under one percent even at fractional-cent prices. Revenue accrues from a configurable platform fee on settlements, an issuer fee at shielded deposit, a relayer fee at withdraw, the cross-chain redemption fee on payment codes, and a 5 percent fee on recurring subscription charges. Because gas is sponsored, RelAI also carries the network fee as a cost of service, which the batch-settlement design keeps bounded.

11.2 The $REL token

RelAI's published tokenomics define a native utility token, **$REL**, with a fixed one-billion supply and deflationary mechanics (a small auto-burn per transaction, buyback-and-burn, and fee redistribution). $REL is positioned as a utility and governance anchor rather than a speculative instrument: it grants relay-fee discounts and fee tiering, governance voting, and premium-tier access (a holding threshold unlocks lifetime access in place of a subscription). The distribution earmarks a majority to treasury with allocations to team, operations, marketing, ecosystem grants, and community incentives under linear vesting. The intent is to tie token value to product usage: the more the marketplace settles, the more fees flow into burn and buyback.

11.3 The adoption funnel

The economic design is a funnel: a **free tier** acquires users and agents; **micro-USDC** payments convert them to paying callers at a fair per-call price; **$REL** holdings unlock discounts, premium access, and governance; and provider **earnings settlement** keeps the supply side profitable even on long-tail, fractional-cent APIs.

---

12. Roadmap

RelAI's privacy and platform work is organized into phases.

- **Faza A (complete): off-chain metadata closure.** Wallet pairs encrypted at rest (AES-256-GCM), authenticated read endpoints, IP scrubbing with daily-salted hashes, ASP-scheduler wallet de-linking, a mandatory RPC proxy to break network correlation, fee-payer relaying for scheduled payouts, and an encrypted client-side vault. This phase removed the metadata leaks that sit around the cryptographic core.
- **Faza B (planned, weeks): on-chain and transport hardening.** Encrypted on-chain payout schedules, stealth recipient addresses (so even the final transfer does not reveal the seller's main wallet), Oblivious HTTP for request privacy, and a dust pool to thicken small-amount anonymity sets.

- **Faza C (research, quarters): trust minimization.** Fully homomorphic encryption for selected flows, a trustless ASP, private information retrieval for witness queries, and a mixnet transport.

In parallel, the path to **mainnet** runs through a multi-party trusted-setup ceremony, verifier immutability, distributed (2-of-3 multisig) ASP authority, and the operational scale work (cursor-based note indexing, sponsored-fee fallbacks) needed to move from devnet to production volumes.

---

13. Conclusion

RelAI makes payment a property of an HTTP request. With x402 as the protocol, a gas-sponsored multi-chain settlement layer, and a zero-knowledge privacy stack that stays compliant by construction, it turns any API into an instant, metered, optionally private service that both humans and autonomous agents can pay for one call at a time. The cryptography guarantees that users transact without leaking identity or amount; the Association Set Provider guarantees that this privacy never becomes a haven for sanctioned funds; and the relayer architecture guarantees that none of RelAI's own infrastructure has to be trusted with a user's secrets. As AI agents become the dominant consumers of online services, a payment fabric that is machine-native, private, and compliant is not a feature: it is the missing settlement layer for the agentic web.

Appendix A: Glossary

- **x402:** HTTP standard for per-request payment using the `402 Payment Required` status code; v1 (header plus URL facilitator) and v2 (structured challenge plus signature) coexist.
- **Facilitator:** a service that verifies and settles an x402 payment on chain.
- **CAIP-2:** a chain-agnostic identifier (`namespace:reference`) used to name networks in challenges.
- **EIP-3009:** an Ethereum standard for off-chain, gas-less USDC transfer authorization, used by RelAI's sponsored EVM settlement.
- **feePayer (Solana):** the account that pays a transaction's fee; RelAI sponsors it so users never need SOL.
- **Commitment:** a Poseidon hash of secret note material, stored as a Merkle leaf; hides the amount.
- **Nullifier:** a value revealed when spending a commitment that prevents double-spends without revealing which commitment was spent.
- **Shielded pool:** a UTXO privacy system over a Poseidon Merkle tree.
- **SPR (Shielded Payment Request):** a private buyer-to-seller payment through a shared shielded pool.
- **Quote:** a seller-issued, Poseidon-committed secret binding an amount and recipient, shared as a `relai:quote:` payload.
- **Payment code:** a short bearer instrument redeemable to any wallet with the amount hidden on chain.

- **Subscription:** an on-chain recurring payment on Solana; the subscriber authorizes once and the program enforces a capped, revocable, per-period pull that RelAI's scheduler executes.
- **ASP (Association Set Provider):** a second Merkle tree of screened, approved deposits; spending requires membership, enforcing compliance without custody.
- **Groth16 / BN254 / Poseidon:** the proof system, elliptic curve, and hash function underpinning RelAI's zero-knowledge layer.
- **ERC-8004 / SATI:** on-chain agent identity and reputation registries (EVM and Solana respectively).
- **Faza:** a roadmap phase; Faza A (off-chain metadata closure) is complete, B and C are planned.

This whitepaper describes a system under active development. On-chain addresses cited are devnet or testnet deployments unless noted; mainnet deployment is gated on the trusted-setup ceremony and operational hardening described in Section 12. Figures and token parameters reflect the current published design and may evolve.