

# The Agentic Internet Stack

Research and Implementation Guide for Autonomous Agents:  
Orchestration, Mission-Bound Authorization, Private Execution,  
Programmatic Payment, and Verification Proofs

August 2026

## Abstract

Autonomous agents encounter an infrastructure problem when they are trusted to execute consequential workflows across organizational boundaries using public internet rails. At that boundary, authority is held in one system, private policy in another, execution evidence in logs or attestations, and payment and settlement somewhere else. No shared state keeps the parties' commitments and the resulting outcome intact. The general missing capability is programmable private application state with a mission-centered orchestration model, portable proofs, and settlement: private rules and inputs can govern a shared state transition without being disclosed, while the accepted claim remains independently verifiable. For agents, that general capability takes the form of verifiable mission state, a durable object that binds identity, bounded authority, commitments, evidence requirements, payment, and outcome. This is what defines the agentic internet stack.

**The key architectural finding is that the Zeko protocol and the o1js TypeScript DSL are best suited to provide this programmable, proof-backed state and orchestration layer for agents and AI systems.** o1js expresses private application logic in TypeScript; the Kimchi zero-knowledge (ZK) proof system turns selected policy and state claims into succinct ZK proofs; the Pickles recursion layer composes accepted proofs into recursive batches; and Zeko efficiently executes and preserves the resulting proof-backed state with immediate transaction confirmation. Ethereum provides the principal Turing-complete decentralized L1 settlement environment. A2A and MCP, enterprise identity and key systems, models and GPU workers, TEEs, SMPC, FHE, zkVMs, zkEVMs and STARK systems, and data-availability systems remain open integration edges, each used where it excels. Native CPU Kimchi proving supplies a portable, flexible, and horizontally scalable baseline for compact claims, while ordinary AI execution remains on GPU-native infrastructure. The result is a general application architecture for agent workflows that preserves authority, privacy, evidence, payment, and outcome as one mission even as specialized systems at the edges change.

# Contents

|   |                                                         |    |
|---|---------------------------------------------------------|----|
| 1 | Executive Summary                                       | 1  |
| 2 | Architecture, Lifecycle, and Deployment Patterns        | 4  |
| 3 | Sovereign Zeko rollups and Ethereum settlement          | 9  |
| 4 | Proving speed, cost, and Zeko capacity                  | 14 |
| 5 | Requirements-driven implementation                      | 16 |
| 6 | Conclusion: programmable private state for agentic work | 18 |
| A | Benchmark protocol and claim taxonomy                   | 19 |
| B | Abbreviations and technical terms                       | 20 |
| C | Technology reference                                    | 21 |
|   | References                                              | 24 |

# 1 Executive Summary

## 1.1 The problem: trust fragments across agentic workflows

The objective of this research and implementation guide is to define a **streamlined, deliberate, interoperable architecture** through which independent agents can discover one another, receive bounded authority, execute private workloads, exchange verifiable evidence, make machine payments, and settle outcomes across organizations and the public internet without exposing mission data.

The individual components already exist. A2A and MCP can connect agents and tools; identity systems can issue credentials; GPUs and TEEs can execute workloads; signatures, ZK proofs, or attestations can provide evidence; stablecoin rails can move value; and Ethereum can settle assets. The failure occurs between those components. As a workflow crosses boundaries, the authority under which it began, the private rules that governed it, the evidence it produced, the payment it earned, and the outcome it may settle are easily reduced to unrelated records under different operators.

The underlying platform requirement is **programmable private application state with portable proofs and settlement**. Private rules and inputs can determine whether a transition is valid; commitments and proven claims can then become shared state that other systems verify and settle. This is a general-purpose capability for finance, identity, markets, enterprise policy, and arbitrary zkApps, not only for agents. Here, “private state” does not imply that every byte is hidden on-chain: witnesses and confidential rules remain private, while the commitments, permissions, and accepted claims needed for coordination become verifiable state.

For agentic work, this general primitive is best organized as **verifiable mission state**: a shared, programmable record of identity, scope, budget, authority, commitments, side-effect policy, evidence, payment, and settlement outcome. It is more than a message envelope or audit log. It is the state against which every participant can determine what was authorized, what was committed, what has been established, and what may happen next without requiring disclosure of the mission’s private inputs or rules.

## 1.2 The missing layer: the Zeko architecture

Zeko is a programmable, proof-backed state and settlement substrate for workflows in which independent parties must accept a state transition governed by private rules without treating one party’s database as the final authority. Agent missions are one application of that substrate. Zeko is therefore the proof-backed state and settlement substrate beneath this orchestration architecture. It is not the scheduler, model host, or general agent runtime. At the platform level, it carries programmable state, proof authorization, recursive batches, and settlement semantics. Zeko preserves the state, authority, policy, evidence, payment conditions, and settlement outcome that counterparties must be able to trust. Existing models, tools, identity systems, and payment rails remain intact at the edges.

**o1js**, the TypeScript SDK built by o1Labs for writing zkApps, lets developers express private, provable application logic in familiar code. Kimchi turns that logic into succinct evidence; Pickles composes proofs into recursive batches; the native Rust verifier makes accepted proofs usable across services and settlement adapters; and Zeko carries the resulting state and settlement semantics. Agent missions are one especially complete application of this model, but the same path supports private finance, identity, policy, markets, and other zkApps. For recurring policy, authorization, and state claims, native **CPU Kimchi proving** is the practical default: broadly deployable, flexible across circuit families, and horizontally scalable across independent receipts. GPUs then target the regular field-arithmetic, FFT/evaluation, and MSM kernels when lower single-proof latency justifies specialized hardware.

The resulting stack is Zeko-centered but deliberately open at its edges. Among Zeko-native protocols

and application primitives, the **Agent Mission-Bound Authorization protocol (AMBA)**, or Agent Mission-Bound Auth for short, constrains delegated authority, while ZK x402 attaches machine-payment context. These components are an initial cross-section of the Zeko application surface, not its boundary: they demonstrate a general substrate that can support a much broader class of workflows whose private rules, accepted state, evidence, or settlement must remain portable across parties. Encryption and key management protect data; TEEs provide low-latency confidential execution; SMPC and FHE serve selected private-computation cases; zkVMs prove general execution; and zkEVM/STARK systems provide EVM-compatible or transparent proof paths. Zeko is the recommended substrate where shared application state, private rules, external verification, or settlement make one operator’s database insufficient; specialist systems remain preferable everywhere their narrower trust model already solves the problem.

Enterprises can deploy this architecture as a sovereign Zeko rollup: a private and permissioned application domain with organization-controlled sequencing, proving, data availability, keys, and governance. Inputs, outputs, application state, and operational artifacts can remain inside the enterprise boundary, while only the commitments and proofs required for external verification and Ethereum settlement leave it. This preserves enterprise control without requiring counterparties to treat the enterprise’s internal database as the final authority.

Zeko’s sequencer confirms a proof-backed transaction as soon as it is received; its rollup state transition proofs, recursive batch proof, and external settlement follow without delaying that confirmation.<sup>[1]</sup> Capacity grows through three complementary mechanisms. **Horizontal scaling** adds independent prover or rollup lanes; **vertical scaling** improves the latency and concurrency of one rollup; and **recursive scaling** uses Pickles to compress proofs and rollup batches before settlement. Throughput therefore becomes the aggregate throughput of the rollup hierarchy rather than the capacity of one sequencer, allowing orders-of-magnitude growth as additional rollups are added. Section 4 shares measured results and the proposed sequencer and GPU work within that model.

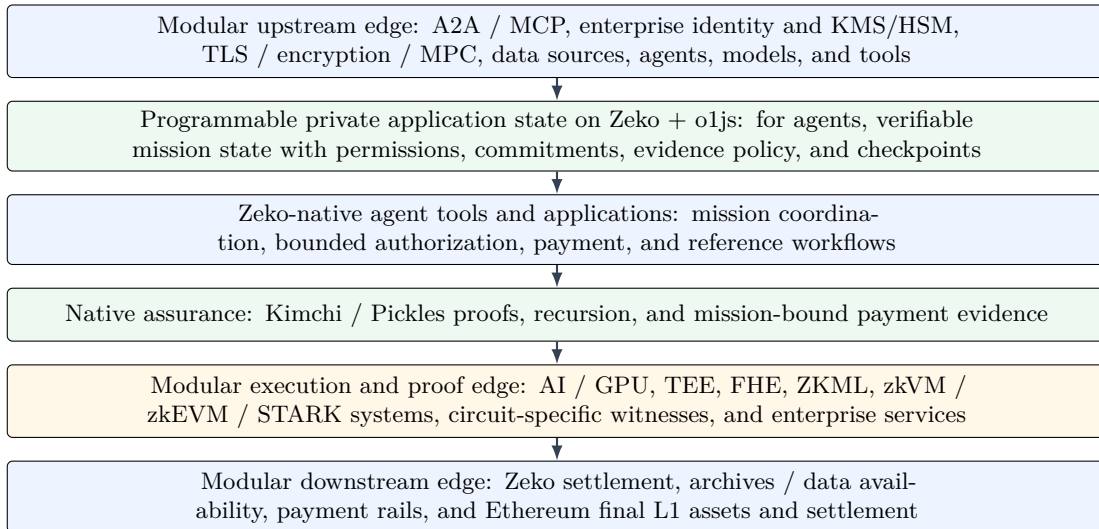


**Figure 1.** Reference architecture: Zeko and o1js supply a general proof-native application-state layer; agent workflows organize that state around the mission.

For qualifying workflows, **Zeko and o1js provide one programming, proof, state, and settlement model**. A team defines private policy and application state in TypeScript, proves only the facts that require cryptographic evidence, and carries the accepted result through recursive batches and settlement. That general model can govern a financial position, identity claim, market rule, enterprise approval, or arbitrary zkApp transition. For agents, the mission binds requester, worker, authority chain, budget, commitments, evidence policy, and settlement rule; as work proceeds, that same state

accumulates receipts, attestations, proofs, and payment authorizations.

This creates a coherent center without closing the system. A2A/MCP, identity and key systems, proprietary data, models, tools, and GPU workers supply inputs and execution upstream. Data availability, archives, accounting, bridges, Ethereum assets, and counterparties consume commitments and settlement events downstream. Because the proof-backed application state survives changes at either edge, teams can establish the state object first—a mission and receipt in the agent case—then add only the privacy, execution, and proof mechanisms that risk justifies.

**This continuity creates a portable, proof-backed source of truth across chains and run-times.** Rather than independently recreating application logic and state on every Ethereum L2, competing EVM L1, agent runtime, or enterprise system, Zeko preserves one canonical application-state and proof model. Each connected environment can consume a proof-backed claim derived from the same accepted Zeko state transition, allowing policies, receipts, commitments, and outcomes to retain consistent meaning wherever they are used. The destination networks and runtimes remain operationally independent, but they reference a common cryptographic truth instead of becoming separate application systems of record. As verification and settlement integrations expand, that same truth can be carried into additional environments without changing the underlying application model.[21, 22, 10, 7]

### 1.3 The mission as the orchestration object

When independent agents work together, the result must remain connected to the authority and terms under which it was produced. The paper calls the shared object that carries those facts **verifiable mission state**. It records who authorized the work, what the worker could do, which inputs and tools were committed, what evidence was promised, when payment may be released, and how the result can be challenged. The *mission receipt* is its portable evidence snapshot:

$$R = H(\text{mission\_id}, \text{requester}, \text{worker}, H(\text{inputs}), \text{policy}, H(\text{code}), H(\text{model}), H(\text{outputs}), \text{evidence}, \text{payment\_state}). \quad (1)$$

The receipt is signed on the fast path, bound to a TEE attestation, a circuit proof, or a zkVM proof when the mission requires stronger evidence, and committed to Zeko when the outcome needs durable protocol settlement. The Zeko-native SantaClawz protocol carries the backend lifecycle and rules; AMBA binds enterprise approval and checkpoints; Magic City integrates the SantaClawz and AMBA protocols into a user-facing internet product; and ZK x402 binds payment authorization to the same object. Externally enforceable assets or final commitments can then settle on Ethereum.[7]

Evidence is selected by the claim, not by a fixed cryptographic ladder. A signed receipt is sufficient for many low-risk missions; attested execution adds confidentiality and environment identity; a Kimchi circuit proves compact policy or state predicates; a zkVM proves general program execution; and Pickles compresses repeated receipts or checkpoints for Zeko settlement. This keeps the common path fast while making stronger evidence available exactly where the mission requires it.

### 1.4 Technology landscape and selected stack

The architecture follows one rule: assign each technology the narrowest role it performs well, then bind consequential outputs to programmable Zeko application state when the workflow requires a shared proof-backed record. In an agent workflow, that shared state is the mission. The comparison asks practical questions: can a component preserve the relevant application semantics, verify the required claim at acceptable cost, compose evidence, support enterprise governance, and connect cleanly to existing systems?

The fit threshold is practical. Conventional enterprise controls remain the efficient center when one accountable organization is accepted as authoritative. Zeko becomes relevant when private rules, cross-party verification, proof portability, or an external asset or remedy make one operator’s database insufficient. It should secure consequential transitions, not every model call, message, or log line. The detailed division of labor appears in Section 2, and the implementation fit test in Section 5.

The relevant technologies are complementary, but they are not interchangeable. Conventional encryption and key management protect stored and transmitted data; TEEs, SMPC, and FHE protect selected forms of computation; application-specific ZK proves compact recurring claims; zkVMs prove general programs; zkEVMs and STARK systems serve EVM-compatible, large-trace, or transparent-proof settings; and Pickles composes many accepted claims into a compact checkpoint. Appendix C gives the full comparison. The selected stack below keeps that entire landscape available without forcing every mission through its most expensive component.

**Table 1.** The selected production stack for qualifying workflows and the boundary of each component.

| Layer                             | Primary component                             | Rationale                                                                                           |
|-----------------------------------|-----------------------------------------------|-----------------------------------------------------------------------------------------------------|
| Agent edge                        | A2A/HTTP plus MCP adapters                    | Connect existing agents and tools without making A2A or MCP the trust root.                         |
| Proof-backed state and settlement | Zeko application state and sovereign rollup   | Keep private rules, permissions, evidence, and settlement attached to accepted state.               |
| Mission applications              | SantaClawz, Magic City, and AMBA              | Coordinate missions, integrate the agent interface, constrain authority, and verify receipts.[8, 7] |
| Fast execution                    | GPU workers, optionally inside Phala CVMs     | Run models and tools at practical latency; add confidential execution when required.                |
| Compact proof                     | o1js/Kimchi                                   | Prove selected policy and state claims without proving an entire runtime.                           |
| Recursive compression             | Pickles                                       | Compress many receipts or checkpoints before settlement.                                            |
| General execution proof           | RISC Zero or SP1 only when needed             | Prove arbitrary execution when an application-specific claim is insufficient.                       |
| Machine payment                   | ZK x402                                       | Bind payment negotiation and authorization to the accepted mission state.[7]                        |
| Settlement                        | Zeko settlement; Ethereum final L1 settlement | Use Zeko for protocol state and Ethereum for external assets, contracts, liquidity, and finality.   |

## 2 Architecture, Lifecycle, and Deployment Patterns

This architecture applies first to consequential workflows that cross organizational boundaries over public internet rails, especially where money, access, compliance, confidential data, or physical outcomes are at stake. These include cross-company procurement and stablecoin commerce; private treasury, credit, and settlement under confidential policy; secure exchange and analysis of confidential data across enterprises; and multi-operator logistics, vehicle, robot, or industrial networks. AMBA bounds authority, native Zeko zkTLS authenticates private external data, Kimchi and Pickles provide portable evidence, and ZK x402 or Ethereum-facing settlement releases value against accepted Zeko state. Execution stays with existing services, GPUs, TEEs, and edge controllers; Zeko keeps authority, evidence, payment, and outcome attached to the workflow.

## 2.1 Agent Mission-Bound Authorization across the lifecycle

**Agent Mission-Bound Authorization (AMBA), or Agent Mission-Bound Auth, keeps a mission’s authority enforceable as it moves through otherwise independent systems.** Enterprise identity establishes the principal, A2A describes agents and tasks, MCP describes tools, and account credentials open particular systems. AMBA answers the remaining operational question: *what may this holder do for this mission, against which resources, for how long, within what budget, and subject to which evidence and settlement conditions?* It converts authenticated approval into a portable, attenuated capability checked at every consequential boundary.[9, 15, 16]

The mission defined in Section 1 becomes a five-stage operating lifecycle—discover, authorize, execute, verify, and settle—with identity, authority, evidence policy, and payment context attached to the same state object throughout.

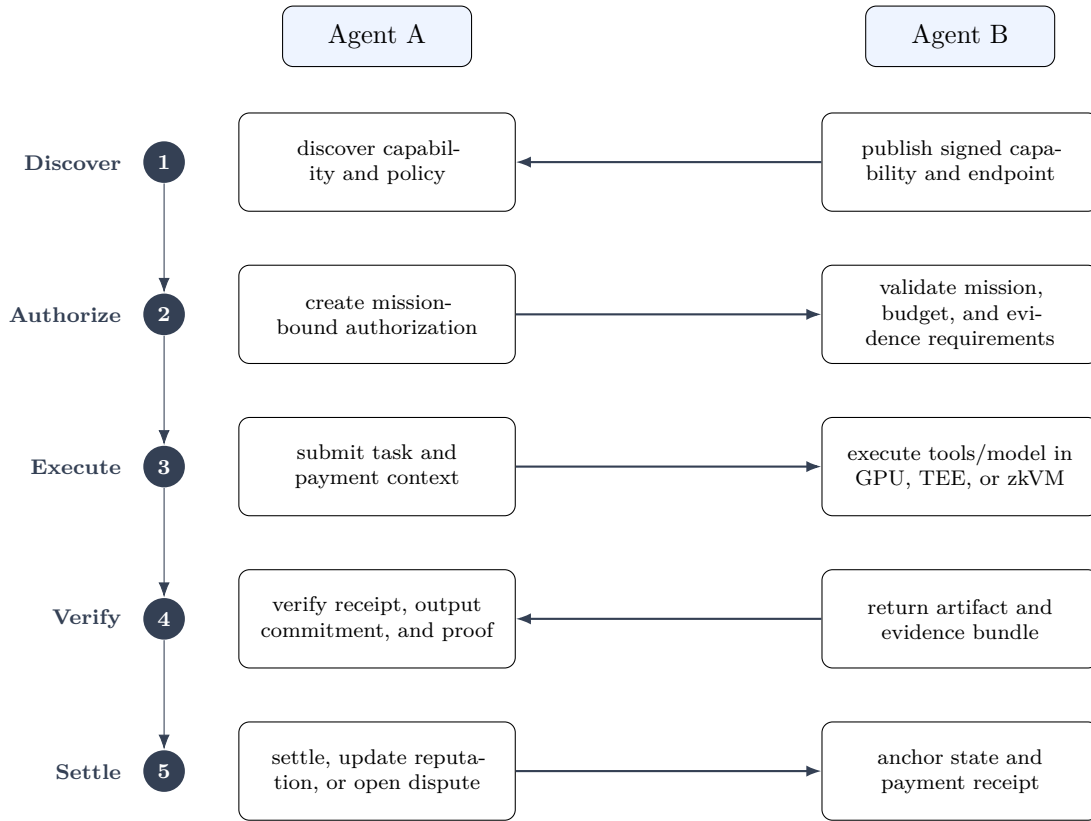


**Figure 2.** The numbered stages show the sequential mission lifecycle. The mission remains a durable object across network disconnects and can be supported by progressively stronger evidence.

### 2.1.1 Control objects and invariants

The architecture uses four related objects. A **mission policy** commits to permitted actions, domains, data scopes, payment rails, maximum spend, expiry, required checkpoints, and receipt requirements. A **mission capability** binds that policy to a mission, agent or runtime, holder key, audience, expiry, unique identifier, and settlement nullifier. A sequence of **boundary events** records the consequential actions actually attempted; each event is signed by the holder and commits to the preceding event. A **portable receipt** commits to the capability, policy, holder, completed trace, payment context, evidence, nullifier, and anchor state.

In compact form, the control relationship is:

$$\begin{aligned}
P &= H(M, \text{actions}, \text{domains}, \text{data}, \text{rails}, \text{spend}, \text{expiry}, \text{checkpoints}), \\
C &= \text{Auth}_I(H(M), H(P), A, K_H, \text{aud}, \text{expiry}, \text{jti}, N), \\
E_i &= \text{Sign}_{K_H}(H(C), H(P), a_i, d_i, r_i, p_i, H(E_{i-1})), \\
R &= H(H(C), H(P), K_H, H(E_1, \dots, E_n), \text{payment}, \text{evidence}, N, \text{anchor}).
\end{aligned} \tag{2}$$

Here  $I$  is the mission authority,  $A$  is the selected agent or runtime,  $K_H$  is the holder key, and  $N$  is a one-time settlement nullifier. The mission authority authenticates the capability; for portable enterprise verification this should use a public verification key advertised through JWKS, while an operator-local MAC may remain a lower-assurance internal fast path. The holder signature prevents a copied capability from becoming a freely replayable bearer token. The event chain makes deletion, reordering, or substitution detectable, and the nullifier prevents the same accepted receipt from authorizing payment twice.

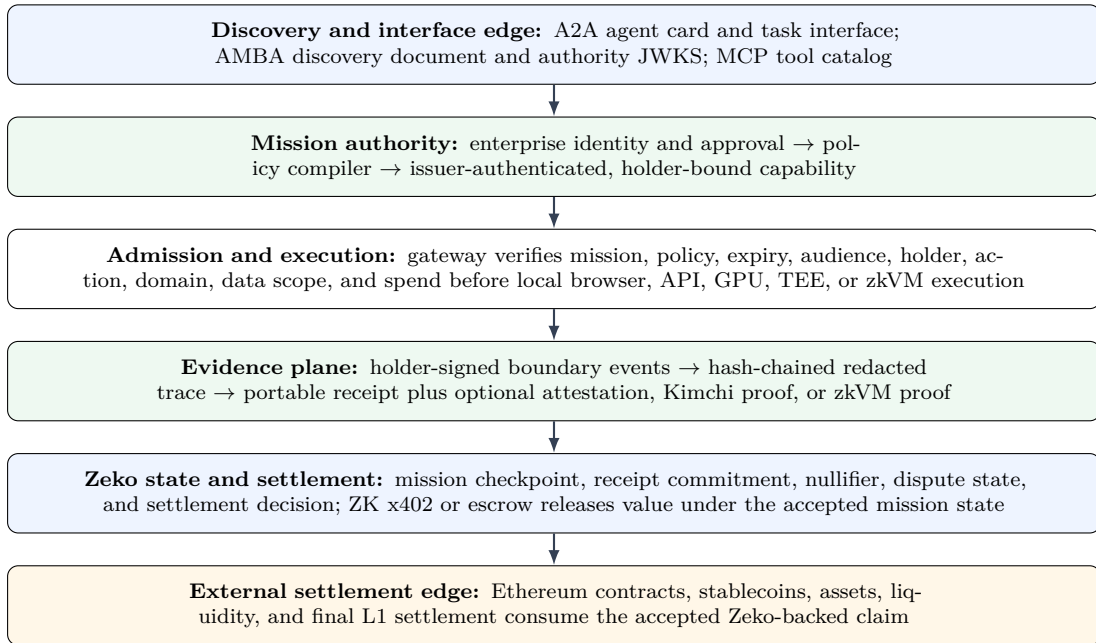


**Figure 3.** Agent Mission-Bound Authorization carries scoped authority from discovery to execution, evidence, and settlement without making the surrounding protocols the trust root.

### 2.1.2 Discovery, admission, and delegation

At discovery time, an agent can continue to publish an A2A Agent Card and expose MCP-compatible tools. AMBA adds a verifier-facing well-known authorization document that advertises the capability, policy, boundary-event, receipt, and registry schemas; issuance and verification endpoints; supported holder-proof schemes; enforcement points; payment rails; and Zeko anchor method. A separate authority JWKS endpoint allows a counterparty to verify issuer signatures without sharing an application session or trusting a private API call. Discovery publishes formats and verification methods, not user permissions or mission secrets.

After the requester selects an agent, the mission service compiles the request, enterprise approval, budget, and data policy into  $P$ . The issuer then creates  $C$  for a particular audience and runtime. On claim, the runtime proves possession of  $K_H$ ; a gateway should reject unsigned checkpoints and a capability presented by any other holder. The capability remains short-lived and least-privileged. Its allowed actions and domains must be explicit, its data scope must identify what may be read or disclosed,

and its payment authority must be bounded independently from ordinary tool access.

Delegated multi-agent work forms a **mission tree** and uses **attenuation**, not delegation by copy. The original authority is the root mission; each branch capability references the mission node that authorized it through `authorizing_mission_id` and satisfies

$$\begin{aligned} scope_{branch} &\subseteq scope_{authorizing}, \\ spend_{branch} &\leq remaining\ spend_{authorizing}, \\ expiry_{branch} &\leq expiry_{authorizing}. \end{aligned} \tag{3}$$

A branch may narrow actions, domains, data, evidence, or time, but may not enlarge them. Branches that delegate further remain internal mission nodes; terminal branches produce leaf receipts. Pickles can recursively fold those leaf receipts and branch checkpoints into a compact root checkpoint, while every branch retains its own holder key and nullifier. Production deployments should also maintain a revocation or status commitment checked before every external side effect. Short expiry and re-issuance are useful defaults, but they do not replace immediate revocation for a compromised holder or cancelled mission.

### 2.1.3 Execution, privacy, and checkpoints

AMBA enforcement belongs immediately before consequential boundaries: private-data release, external API mutation, agent hiring, payment offer, payment authorization, final submission, and receipt acceptance. The runtime maps a proposed operation to a canonical mission action, verifies the capability and latest status, checks the action, domain, data scope, expiry, and spend policy, and either proceeds, pauses for approval, or fails closed. The mission-aware gate admits each MCP call or A2A task only after those checks; neither interface by itself conveys payment, credential, browser, or settlement authority.

Execution can remain local or use the environment best suited to the workload. A browser helper can operate inside the user’s authenticated profile; a GPU worker can run a model; a TEE can protect remote inputs and code; and a zkVM can prove arbitrary execution when required. In each case, the worker exports only the evidence promised by the mission. The reference browser profile, for example, commits to session, tab, domain, page-state class, safe-next-action decision, stop reason, and checkout checkpoint without exporting URLs, selectors, field values, card data, cookies, or search text.

After each meaningful boundary, the holder signs  $E_i$ . The signed payload binds the mission and policy commitments, action, target-domain and resource commitments, payment-context digest, idempotency key, and prior event hash. Private details remain with the worker; a public or counterparty-facing trace contains hashes, normalized event types, timestamps, and proof metadata. Hard stops should include login or MFA challenges, payment authentication, policy conflict, budget exhaustion, an unknown domain or action, and any final action whose policy requires visible approval.

### 2.1.4 Receipt, proof, and settlement

The receipt is the transition from operational logging to portable evidence. A verifier can check that the receipt refers to the issued capability and policy, the trace is complete and holder-bound, the payment context is unchanged, and the nullifier has not been consumed. The evidence attached to that receipt remains proportional to the claim: an issuer and holder signature for the fast path; a TEE attestation for a measured confidential worker; a Kimchi proof for recurring policy, authorization, trace, spend, or state predicates; or a zkVM proof when arbitrary program execution itself must be established. Pickles can recursively compose leaf receipts and branch checkpoints into a root checkpoint without changing the mission interface.

Zeko stores the accepted application-state transition rather than the private trace. A registry or mission zkApp can commit to the proof statement hash, receipt or batch payload digest, current root, nullifier, and settlement decision. ZK x402, credits, escrow, or a stablecoin adapter can then condition release on that accepted state; Ethereum contracts and assets remain downstream consumers when final L1 settlement or broad liquidity is required. A dispute references the same mission, policy, trace, and receipt commitments, avoiding a second audit record assembled after the fact.

**Table 2.** Agent Mission-Bound Authorization checks across the agent lifecycle.

| Lifecycle stage     | AMBA artifact or invariant                                                                     | Relationship to the surrounding stack                                                                     |
|---------------------|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| Discovery           | Authorization document, schemas, issuer keys, verification and anchor endpoints                | Complements A2A discovery and MCP catalogs without publishing mission secrets.                            |
| Mission formation   | Policy commitment covering actions, domains, data, payment, expiry, checkpoints, and evidence  | Compiles enterprise IAM approval and application policy into a portable mission object.                   |
| Claim and admission | Issuer-authenticated capability bound to a runtime holder key, audience, expiry, and nullifier | Prevents an agent listing, OAuth token, or copied bearer credential from becoming unlimited authority.    |
| Execution           | Pre-boundary policy decision plus holder-signed, hash-chained event                            | Gates MCP tools, A2A delegation, local browsers, APIs, GPUs, TEEs, and zkVMs at the point of consequence. |
| Evidence            | Redacted trace and portable receipt, optionally carrying attestation or proof                  | Keeps private inputs local while allowing counterparties to verify the promised claim.                    |
| Settlement          | Zeko checkpoint or registry anchor, unspent nullifier, and release or dispute decision         | Connects ZK x402, escrow, stablecoins, and Ethereum settlement to the exact accepted mission state.       |

## 2.2 Implementation evidence: SantaClawz and Magic City

For agents working across the public internet, mission-bound authority must remain intact as discovery, coordination, execution, evidence, and payment move between independently operated systems.

SantaClawz implements the mission lifecycle at the protocol layer, coordinating discovery, mission creation, role assignment, work handoffs, evidence, and payments. Magic City integrates SantaClawz and AMBA into a user-facing internet experience. Together they demonstrate that mission-bound authority can remain coherent across holder binding, signed boundary events, portable receipts, verification, and Zeko-compatible settlement inputs.

The SantaClawz Workshop is an optional private, role-based inter-agent messaging and verification workspace that routes encrypted workflows while retaining only policy-permitted mission metadata and receipt commitments. The significance is architectural rather than product-specific: agent interfaces and collaboration surfaces can evolve without changing the underlying mission, authority, proof, payment, or settlement model.[9, 8, 7]

## 2.3 Execution, privacy, and evidence

Each component retains a bounded role. A2A and MCP connect agents and tools; signed capabilities provide the low-risk authority path; TEEs provide confidential execution; zkVMs prove arbitrary execution; oljs and Kimchi prove compact application claims; Pickles composes accepted history; Zeko preserves mis-

sion state; and Ethereum supplies external assets and composability. This division lets teams strengthen only the claims that require it. Appendix C records the qualitative technology comparison.[1, 14, 7]

Evidence composes in progressively stronger layers around the same accepted mission state. Signed Zeko receipts provide a durable, low-overhead state reference; TEE attestations add confidential execution; Kimchi proves compact private policy and state predicates; zkVM receipts prove arbitrary program execution; Pickles compresses accepted receipts into recursive checkpoints; and Ethereum settlement connects the resulting state to external assets and contracts. Because every artifact is bound to the same mission, stronger assurance can be added without replacing the agent, model, tool, or execution runtime.

A TEE supplies low-latency confidential execution and attests to the identity and configuration of a measured environment; it does not prove arbitrary AI correctness. Within the mission lifecycle it is an optional execution and evidence lane. The sovereign Zeko deployment pattern inside a Phala CVM, including key release and operating controls, is addressed in Section 3.3.

### SMPC as a private-inference connector

SMPC serves as a selective private-inference connector when prompts must remain private from a model provider, non-colluding parties are available, and statistical assurance is sufficient. The preprint combines two-party SMPC with logit fingerprinting and noise; its Llama-2-7B result is about  $15\times$  faster than zkLLM, not ordinary GPU inference.[27]

The worker returns a **private inference verification receipt** committing to the model, prompt, output, result, and trust policy. AMBA binds it to the mission and ZK x402 gates payment; Kimchi proves the receipt-policy predicate; Pickles/Zeko aggregate and checkpoint receipts. Its assurance is bounded: the noise-based protocol assumes non-collusion and statistical classification, not ZK soundness. Basic fingerprinting admits a subset attack; FHE is a costlier single-provider alternative.[27]

## 3 Sovereign Zeko rollups and Ethereum settlement

For an enterprise, sovereignty means owning a **cryptographically enforced application-state domain**. That domain can be fully private at the application-data layer: admission is permissioned; inputs, outputs, state payloads, and artifacts are encrypted; data availability and archives are restricted; and key release remains under enterprise policy. Commitments and proofs can still cross the settlement boundary without disclosing the underlying data. The organization controls membership, permissions, sequencing, data availability and archives, privacy policy, proving operations, governance, and recovery while retaining an external L1 settlement path. A single institution can operate an application-specific rollup; a consortium can share its governance and consensus policy; and a larger rollup estate can connect siloed, application-specific domains through common proof and settlement interfaces. Zeko documents custom rollups with selectable DA, archivation, bridging, and operator controls.[4, 5] The organization chooses the operating model, while valid state transitions remain enforced by the circuit, proof system, DA policy, and settlement layer.

Ethereum is selected as the open-internet destination for assets and final L1 settlement, though Zeko’s modular design supports additional settlement destinations. The *Ethereum ZK Rollup Framework: Technical Litepaper* specifies the corresponding verification and bridge architecture.[10] Separately, ZK x402 is live for machine-payment use on Zeko and with EVM payment rails; it is an application-payment protocol, not the native chain-proof settlement path.[13]

### 3.1 What an enterprise actually owns

A sovereign deployment gives the organization a **proof-native control plane**: the policy through which missions are admitted, authorized, sequenced, proved, and settled. The sequencer and related services put identity, authority, budget, data classification, and side-effect policy directly around the write path.[1, 4] This is more than dedicated hosting. It is a programmable institutional boundary that can still produce evidence for customers, counterparties, auditors, and public settlement contracts.

**Table 3.** Enterprise control surfaces in a sovereign Zeko rollout.

| Control surface                     | Enterprise design authority                                                                                           | What it enables                                                                                                            | Implementation condition                                                                                         |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| Admission and delegation            | Participant registry, tenant namespaces, mission schema, AMBA, approval quorum, budget, and side-effect limits        | A controlled network in which agents act only within a named, reviewable mandate                                           | Commit material authorization facts to mission state or cover them with the governing signature or proof.        |
| State transition policy             | zkApp preconditions, account permissions, verification keys, custom-token rules, and circuit versioning               | Fine-grained control over what balances, commitments, status roots, or permissions may change                              | Treat circuit and verification-key governance as part of the security perimeter.                                 |
| Execution and proving               | Enterprise sequencer, prover fleet, GPU workers, and, where warranted, Phala CVMs                                     | Performance isolation, regional operation, model and tool pinning, confidential workers, and predictable service ownership | TEE attestation complements, but does not replace, proof soundness, key governance, or image review.             |
| Data availability and audit history | DA provider, archive relay, retention period, artifact store, encryption, and key-release policy                      | A tailored balance of recoverability, auditability, data residency, and access control                                     | Safe recovery requires durable data availability, key recovery, and authorized-disclosure procedures.            |
| Governance and continuity           | Multisignature change control, emergency pause authority, upgrade process, backups, monitoring, and incident response | Controlled upgrades, policy changes, and a clear response to a compromised worker or circuit                               | Separate narrow emergency powers from ordinary operations and define sequencer failover and censorship response. |

The mission receipt carries enterprise approval, delegation, payments, commitments, and evidence into the accepted state transition rather than leaving policy as an off-chain assertion.

### 3.2 A private and permissioned yet externally verifiable control plane

Full application-data privacy and external verifiability are compatible. A sovereign rollout can restrict who enters and what they may do, encrypt its inputs and outputs, keep application state and artifacts inside an enterprise-controlled data boundary, and still expose cryptographic evidence about accepted state transitions. Membership, KYC/AML, tenancy, limits, and delegated authority remain application policy; consequential admission decisions should be signed or proof-backed and recorded in mission state or evidence. SantaClawz can carry the backend lifecycle and rules, AMBA the approvals and checkpoints,

Magic City the integrated user-facing internet experience, and ZK x402 the payment context, all within the same Zeko state model.

The privacy protections are cumulative: a Kimchi witness hides data from the verifier; client-side proving can keep plaintext from the operator; encrypted inputs and outputs protect the application boundary; a server-side prover can pair encryption with a TEE where the operator must not inspect plaintext; and encrypted or restricted DA controls reconstruction. In this configuration, counterparties and Ethereum contracts verify the accepted claim while the underlying mission data remains private.[10]

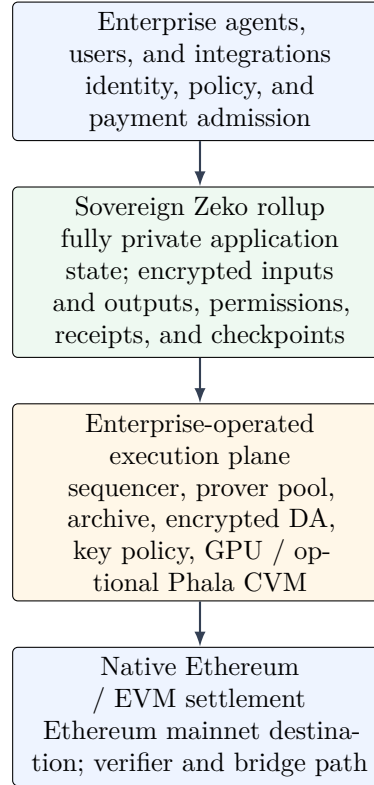


**Figure 4.** A sovereign Zeko deployment keeps application data inside a fully private, enterprise-operated control and execution domain while exporting only the commitments and proofs required for native Ethereum/EVM settlement.

The boundary is deliberate. The sovereign rollup can remain fully private at the application-data layer: encrypted inputs, outputs, state payloads, detailed artifacts, and operational logs stay inside the enterprise-controlled execution and data-availability domain. Ethereum receives only the commitments and proofs needed to enforce the accepted claim; it does not require the underlying mission data to be published.

### 3.3 Operational design considerations for sovereign deployments

Sovereign operation gives regulated organizations explicit control over the infrastructure they are already expected to govern. The implementation plan should therefore name the proof-soundness assumptions, L1 consensus route, data needed for reconstruction and exits, sequencer availability policy, key custody, backups, monitoring, and incident response.[3] Authenticated submission logs, an inclusion policy, failover, a governed emergency pause, and a documented route to the latest available state make those controls auditable rather than implicit.

**Attested sovereign execution.** A TEE can run a private agent, model, sequencer, or prover and

use remote attestation to release keys or authorize mission execution only to a measured image. Zeko documents sovereign-rollup deployment on a Phala CVM; Phala documents programmatic attestations that bind configuration and measurements to TEE hardware.[6, 17, 18] A production deployment adds attested image review, secret release, key rotation, and a receipt-to-measurement audit trail. This is **documented** confidential execution, not proof of arbitrary AI correctness.

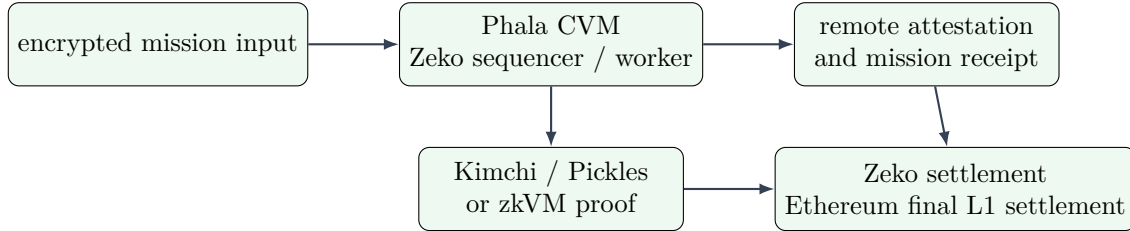


**Figure 5.** Confidential execution and cryptographic evidence occupy complementary assurance lanes: attestation protects the measured environment, while proofs establish explicit protocol or execution claims.

Sequencer selection is a bounded L1 application choice. A deployment can begin with an enterprise-operated sequencer for predictable latency and accountability, then add an auction, committee, or fair-selection mechanism where the governance model calls for decentralization. That change affects ordering, liveness, trust distribution, and potentially latency, but it does not require redesigning the mission schema, oljs application logic, Kimchi/Pickles proof path, or settlement interface. Modular DA similarly lets public, encrypted, or restricted data policies be paired with explicit recovery and disclosure procedures.[5]

### 3.4 Ethereum/EVM extension and state-proof portability

Ethereum/EVM extends Zeko into the smart-contract, wallet, stablecoin, asset, and counterparty environment that internet-facing agent services need. Zeko batch commitments and proofs become Ethereum-verifiable without reimplementing mission logic in Solidity. High-frequency and private application state remains on Zeko, agent execution remains in the appropriate runtime, and Ethereum supplies external assets, composability, bridging, and final L1 settlement.[20, 10]

**State-proof bridging creates a portable, proof-backed source of truth across chains and runtimes.** Zeko preserves one application-state and proof model, then exports accepted roots and claims to destination verifiers. Policies, receipts, commitments, and outcomes can therefore retain consistent meaning as settlement adapters change. This reduces semantic and deployment fragmentation; it does not eliminate fragmented liquidity, message latency, destination finality, or bridge risk. Each destination still needs an audited verifier or proof wrapper, data-availability binding, asset handling, and a recovery path.[22, 21]

ZK x402 is a separate application rail. It attaches HTTP-native payment negotiation and proof-oriented payment authorization to a mission.[13]

*The proof is the truth; the truth is portable with Zeko.*

The final goal is a destination-verifiable Zeko state root on which applications, contracts, and assets can act without recreating the application or treating a bridge operator as the source of truth. Ethereum’s SP1/Groth16 and Solidity path establishes the first EVM adapter; its verification design is broadly reusable, while deployment, asset handling, recovery, and security validation remain destination-specific.

### 3.5 Ethereum verification, bridge security, and performance

The *Ethereum ZK Rollup Framework: Technical Litepaper* defines the verification path:

Kimchi/Pickles → Rust verifier → SP1 proof → Groth16/BN254 → Solidity verifier.

SP1 proves that the Rust Kimchi verifier accepted the Zeko batch, Groth16 makes that result efficient to verify in Solidity, and commitment proofs bind the Zeko state transition to Ethereum’s Keccak and EIP-4844 KZG environment. Ethereum verifies the compact result rather than executing Kimchi natively.[10, 19]

The public settlement record supplies a concrete operating data point: the Zeko-to-Ethereum path completed its SP1/Groth16 proof-and-wrapping request in **under 5 minutes**. The request charged a total fee below \$1.[11, 12] This is the prover-network portion of one observed batch; Ethereum gas, DA, bridge, and operational costs remain separate, while the batch cost is amortized across the accepted transitions it represents.

The bridge should release assets only against withdrawals included in an Ethereum-accepted Zeko root, and should credit Zeko only after the corresponding Ethereum deposit is bound into rollup state. Assurance therefore depends on the verifier, bridge contracts, proof wrapper, DA binding, and key governance as one audited system. Performance must likewise be measured end to end: faster inner Kimchi proving does not by itself accelerate zkVM verification, Groth16 wrapping, blob preparation, or Ethereum submission.[10]

### 3.6 Deployment profiles and the decision to settle externally

Sovereign deployments can take several practical forms. The question is not simply whether to be private or public, but which claims must be independently enforceable outside the enterprise boundary.

**Table 4.** Sovereign-rollup profiles for enterprise agent systems.

| Deployment profile                 | Recommended sovereign configuration                                                                                                                                       | When Ethereum settlement adds value                                                                                                                               |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Regulated internal agents          | Permissioned mission admission, enterprise archive and retention policy, GPU or Phala workers, private artifacts, and Kimchi proofs for the policy claims that merit them | Usually periodic or deferred; use it for external treasury movement, audit anchoring, or settlement with outside counterparties rather than each internal action. |
| Consortium finance or supply chain | Shared governance over sequencer and DA policy, named participant identities, approval thresholds, portable mission receipts, and independently recoverable data          | Use it for stablecoin escrow, net settlement, collateral, or a neutral final state that no consortium member controls alone.                                      |
| Customer-facing paid agent service | Public API edge with Zeko-backed mission receipts, ZK x402 payment context, scalable worker fleet, and selective proof or attestation                                     | Use it where customers pay or receive USDC/other Ethereum assets, require wallet interoperability, or need a contract-enforced refund or escrow rule.             |
| High-assurance autonomous treasury | Narrow mission scopes, multi-party approval for privileged actions, circuit-backed policy checks, attested key release, and recursive checkpoints                         | Use it to settle externally enforceable asset movements only after the mission’s authority, policy, and evidence requirements are met.                            |

The implementation order is practical. First define the admissible mission, privileged actions, data

classes, recovery rights, and external-settlement conditions. Then deploy the sovereign control plane with the sequencer, prover, archive, and DA policy those decisions require. Add attested execution and private artifact handling where a server must process sensitive material. Enable a bridge or external settlement wrapper only after the enterprise can reproduce state, recover from operator failure, and verify its policy circuit. The acceptance question is simple: **can a counterparty independently verify the exact mission claim and obtain the promised remedy if the operator is unavailable?**

## 4 Proving speed, cost, and Zeko capacity

Zeko performance must be read at three distinct layers. A single o1js/Kimchi proof establishes *application-proof latency*; a sequencer and parallel prover deployment establish *rollup transaction throughput*; and Pickles plus batch settlement determine how efficiently accepted state is compressed and carried to an external settlement layer. This paper tested baselines of 2.634 seconds for one unchanged 65,526-gate application proof and 4.9–5.5 transactions per second for sequencer inclusion in the tested deployment. Rollup throughput is bounded by its slowest required stage: sequencing, proving, data availability, or recursive batch production. Horizontal and vertical scaling increase one or more of these capacities, while recursive scaling reduces the downstream verification and settlement required per accepted transition. Application-proof latency remains a separate measurement.

### 4.1 Measured application-proof latency

The first benchmark isolates one application proof. Proof **latency** measures how long that statement takes; proof **throughput** measures how many comparable jobs a fleet completes. The local Rust Kimchi benchmark fixes the statement, keeps the prover index resident, and verifies every proof so that the comparison remains like for like.<sup>[28]</sup> The reporting protocol is summarized in Appendix A.

**Table 5.** Same-statement native Kimchi application-proof benchmark.

| Workload                             | Configuration                                     | Result                                                      | What it measures                                                                                      |
|--------------------------------------|---------------------------------------------------|-------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| 65,526-gate generic Kimchi statement | native Rust, warm resident index, 8 Rayon workers | <b>Measured</b><br>2.634 s median prove; about 75 ms verify | One application-specific policy, authorization, state, and commitment proof on the local Apple M3 CPU |

The 65,526-gate statement is a useful stand-in for one complex enterprise-agent zkApp method proof: a private authorization or policy decision bound to mission state, budget or settlement conditions, and input/output commitments. It does not measure the sequencer’s transaction-SNARK base proof, merge proof, commit proof, or end-to-end inclusion rate.

**What the 2.634-second number means.** This is the fastest verified local result for the unchanged statement: one resident native Rust prover, a warm prover index, and an eight-worker Rayon pool. Those workers parallelize stages *inside one proof*; they are not independent sequencer provers serving a transaction queue. The result is therefore a demonstrated application-proof latency baseline, not a Kimchi-wide limit or a rollup-capacity claim.

**Why this balance works.** o1js lets teams express application-specific mission policies in TypeScript; Kimchi proves the resulting compact predicate instead of re-proving a general-purpose runtime. The design preserves private inputs, succinct verification, and Pickles recursion while allowing new mission types to become new circuits. Its tradeoff is deliberate: programmability lives at the program-and-circuit boundary, while ordinary model and tool execution stays off-chain on the hardware best suited to

it. CPU proving supplies a portable baseline, and independent prover fleets increase throughput without changing application semantics.

**Future work.** A device-resident Kimchi backend would move Pasta-field arithmetic, FFT/evaluation, and MSM kernels to the GPU while leaving witness scheduling, transcripts, proof assembly, Pickles orchestration, and submission on the CPU. Its 0.9–1.3-second target applies to the same 65,526-gate application proof, not rollup TPS. The first claimable result must match the CPU circuit, verification key, transcript, and warm-state conditions and verify with the unmodified Kimchi verifier. Until then, the range remains an engineering target rather than a measured result.[28]

## 4.2 Measured rollup throughput and the scaling path

Zeko does not wait for blocks: the sequencer instantly includes transactions as they are received, providing immediate confirmation while proof completion, recursive batch production, and external settlement proceed asynchronously.[1]

**Horizontal scaling** adds independent prover lanes up to the single-rollup ceiling, then adds sovereign Zeko rollups for partitioned tenants or workloads. The measured rollup baseline is **4.9–5.5 transactions per second** for proof-backed transaction inclusion in a tested sequencer and parallel-prover deployment. This is a horizontal-scaling result: independent transaction-SNARK jobs were distributed across prover capacity while the sequencer preserved ordered state.[1] Under the current sequencer design, **20–30 TPS is the practical single-rollup scaling asymptote**. [1, 11] Each rollup supplies its own sequencing, state, DA, and proving capacity; shared bridge, batch-aggregation, and settlement services remain aggregate limits. Horizontal scaling raises throughput but does not reduce the latency of one unchanged proof.

**Vertical scaling** improves one rollup. Proposed sequencer multithreading would parallelize admission, witness preparation, non-conflicting execution, and prover dispatch while preserving final state order. Faster base, merge, and commit proving could also raise proof-side capacity, but any resulting increase in rollup throughput must be measured across the complete sequencer, prover, DA, and recursive-batch pipeline. This remains ongoing engineering work. With sequencer multithreading, optimized base, merge, and commit proving, sufficient DA and batch-aggregation capacity, and GPU acceleration where proving remains on the critical path, an optimized Zeko rollup can scale well beyond the current **20–30 TPS** single-rollup range. Aggregate capacity can scale further through additional sovereign rollups.

**Recursive scaling** uses Pickles to aggregate accepted proofs and lower-level rollup batches before DA publication or Ethereum settlement. This is where aggregate capacity moves beyond incremental single-rollup gains: independent sovereign rollups execute in parallel and recursively compress their batches into a small settlement footprint. TPS therefore becomes the aggregate throughput of the rollup hierarchy rather than the capacity of one sequencer, allowing orders-of-magnitude growth as additional rollups are added. The system capacity is elastic rather than fixed to one rollup. Recursion reduces what external systems must verify and settle.

Conceptually, aggregate capacity and recursive batch density follow:

$$T_{\text{system}} \approx \sum_{r=1}^R T_{\text{rollup},r} \quad N_{\text{batch}} \approx T_{\text{system}} \Delta t_{\text{batch}}.$$

Here,  $R$  is the number of rollups,  $T_{\text{rollup},r}$  the throughput of rollup  $r$ ,  $\Delta t_{\text{batch}}$  the interval represented by a recursive batch, and  $N_{\text{batch}}$  the accepted transitions it contains. Horizontal scaling raises  $R$ , vertical scaling raises  $T_{\text{rollup},r}$ , and recursion aggregates the resulting  $N_{\text{batch}}$  into a compact settlement object. This model describes capacity and batch density, not total cost. Operating economics also include sequencers, provers, DA, archives, monitoring, redundancy, bridging, and L1 settlement.

## 5 Requirements-driven implementation

Implementation begins with one economically consequential state transition and the counterparty that must accept it. The smallest qualifying deployment adds a versioned mission receipt and Zeko state transition to existing identity, interface, and execution systems. Additional privacy, proof, recursion, payment, or settlement components are selected only when the workflow requires them.

### 5.1 Select the minimum assurance set

Begin with the simplest trust model that satisfies the workflow. Conventional databases, IAM, KMS and HSM systems, TLS, signed logs, and ordinary contractual controls remain the efficient choice when one accountable organization controls the workflow, counterparties accept its records, and no private predicate or external asset depends on an independently accepted result. TEEs add confidential execution when hardware attestation is sufficient; zkVMs prove general computation when a compact application circuit is not.

Zeko becomes compelling when independent parties need shared state without accepting one operator’s database as authoritative; a private rule must govern an externally verifiable outcome; payment, collateral, escrow, or another remedy depends on an accepted claim; operator failure or adversarial behavior is in scope; or the same proof must compose across workflows and settlement domains. Transaction volume can amortize the infrastructure, but the decisive variables are cross-boundary trust, value at risk, confidentiality, proof portability, and enforceable remedy.

The decision turns on five questions: Is one operator accepted as authoritative? Must authority survive across systems or organizations? Which data must remain private? Which claim must an external party verify? Does an external asset or contractual remedy depend on the result?

**Table 6.** Minimum architecture selected by requirement.

| Requirement                                                | Recommended addition                                                                                                                       | Decision rule                                                                                                                                              |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| One trusted operator; signed logs are sufficient           | Existing database, IAM, and audit controls                                                                                                 | Zeko is unnecessary when no external party, asset, or remedy depends on an independently accepted state transition.                                        |
| Independent parties need one accepted application state    | Zeko application state plus a signed mission receipt                                                                                       | This is the core Zeko case: no participant must accept another party’s database as the final authority.                                                    |
| Delegated authority must survive across agents and tools   | Agent Mission-Bound Authorization (AMBA)                                                                                                   | Add AMBA when scope, holder identity, budget, expiry, side effects, and checkpoints must remain attached to the mission across system boundaries.          |
| Inputs, outputs, and application state must remain private | Sovereign Zeko rollup with encryption, controlled keys, and private or restricted DA; add a Phala CVM when a server must process plaintext | Encryption and the sovereign data boundary provide privacy; attestation is needed only when execution must be trusted inside a remote confidential worker. |

| Requirement                                                                       | Recommended addition                                                                                 | Decision rule                                                                                                                    |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| A private rule must govern an externally verifiable result                        | o1js/Kimchi proof attached to Zeko state                                                             | Use a compact application-specific proof for authorization, budget, eligibility, provenance, or another precise state predicate. |
| Arbitrary program execution itself must be proven                                 | SP1 or RISC Zero receipt committed to the mission                                                    | Use a zkVM for general execution; keep Kimchi for compact application and settlement claims.                                     |
| Many proofs or receipts must become one checkpoint                                | Pickles recursion and a Zeko checkpoint                                                              | Add recursion only when aggregation volume justifies its proving and operational cost.                                           |
| Machine payment, escrow, or an externally enforceable asset depends on the result | ZK x402 for mission-bound payment; Ethereum final L1 settlement for assets, contracts, and liquidity | Settle externally when a counterparty needs an on-chain remedy or asset movement, not for every internal mission update.         |

The output of this table should be a one-page assurance profile naming the system of record, authority issuer, private-data boundary, externally verified claim, payment or asset remedy, and settlement destination. That profile is the implementation contract between business, security, and engineering teams.

## 5.2 Minimum deployable pilot

A useful pilot produces five concrete artifacts: a **mission schema** for one state transition; an **authority path** identifying the issuer, holder, scope, expiry, and revocation rule; an **execution adapter** for the existing model, agent, tool, or enterprise service; a **portable receipt and verifier**; and a **Zeko transition** that accepts or rejects the receipt and records the resulting state. AMBA is added when authority is delegated; SantaClawz and Magic City can supply the agent lifecycle and interface; ZK x402 is added when payment belongs inside the accepted mission state.

The end-to-end acceptance test should show that an authorized transition succeeds; an expired, out-of-scope, altered, or replayed receipt fails; private inputs do not appear in externally visible state, logs, or receipts; an independent service can verify the accepted receipt; and payment or another remedy is released only after the required checkpoint. The first pilot should add only the evidence lanes the assurance profile requires: signature, TEE attestation, Kimchi proof, zkVM proof, or a justified combination. Pickles aggregation and Ethereum settlement enter the pilot only when volume or an external remedy makes them part of the requirement.

## 5.3 Production decision gates

1. **Fit and claim.** Name the external verifier, the exact transition it must accept, the value at risk, and the remedy for failure. If the relevant parties accept the operator's database and signed logs, use the simpler system.
2. **Trust and privacy.** Record who may see plaintext, who controls keys and upgrades, which components are trusted, which claim is signed, attested, or proved, and how authorization is

revoked. This becomes the review boundary for the circuit, worker image, verifier, and operating policy.

3. **Capacity and economics.** Benchmark the exact workflow at expected concurrency, including tail latency, memory, failures, retries, proof throughput, and batch-settlement time and cost. Use the measured capacity model to size prover slots, then decide whether recursion, additional rollups, or Ethereum settlement is justified.

A successful pilot leaves behind a versioned schema and, where applicable, circuit; an assurance profile; a reproducible benchmark; an operating and recovery runbook; an independently verifiable receipt; and an explicit settlement decision. Together, these artifacts support an informed deployment decision and give the engineering team a durable foundation for expansion.

## 6 Conclusion: programmable private state for agentic work

This research and implementation guide identifies the missing layer of internet infrastructure for consequential cross-party workflows: **programmable private application state with portable proofs and settlement**. The Zeko protocol and o1js DSL provide that layer: private rules can govern an accepted state transition without disclosure, and the resulting claim remains verifiable across systems. For agents, the durable state object is the **mission**, which binds authority, commitments, evidence, payment, and outcome. Zeko is warranted when those facts cannot depend on one operator’s database; otherwise conventional enterprise controls remain the simpler choice.

The selected architecture is deliberately modular. o1js defines application logic, Kimchi proves compact private claims, Pickles composes them, and Zeko preserves the resulting state and settlement semantics. Identity, A2A/MCP, models, GPU or TEE execution, SMPC, FHE, zkVMs, and data availability remain specialized edges; Ethereum supplies external assets, contracts, liquidity, and final L1 settlement. This division keeps high-frequency and private work off L1 while making accepted claims portable.

The benchmark evidence supports practical deployment, but the larger result is architectural. Once private rules, delegated authority, evidence, and payment can travel as verifiable state, enterprises can let software act across organizational boundaries without exposing the data that makes the work valuable or accepting another party’s database as final authority. Zeko makes that state programmable and portable while execution systems and settlement destinations remain modular. This is the practical foundation of the agentic internet: agents that can carry bounded authority, private data, value, and verifiable outcomes across the public internet rails.

## A Benchmark protocol and claim taxonomy

The report distinguishes empirical results from architectural and engineering claims so that readers can compare systems without mistaking a target for a measurement. A result should inherit the strongest status justified by its evidence, not by the maturity of the surrounding product.

**Table 7.** Claim-status vocabulary used throughout the report.

| Status                             | Required basis                                                                                                      | Appropriate interpretation                                                                                                              |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Measured                           | Reproduced in the stated workspace with verification enabled and the workload identified                            | Evidence about that implementation, hardware, configuration, and statement; not a universal protocol limit.                             |
| Observed                           | Recorded in a supplied or cited test configuration but not independently reproduced in this workspace               | Evidence about the disclosed configuration and workload; suitable as an empirical anchor when its provenance and units remain explicit. |
| Derived                            | Calculated from measured or observed inputs using a stated formula and assumptions                                  | A conditional capacity or cost result, not an independently measured outcome.                                                           |
| Documented                         | Supported by current official documentation or a cited public implementation                                        | An available or described capability whose operational properties still depend on deployment and configuration.                         |
| Prospective:<br>proposed or target | A design or integration path, or a bounded engineering objective, whose production result has not yet been measured | A reviewable architecture or planning range that must be validated before it is represented as deployed performance.                    |
| Open work                          | A required provider, circuit, verifier, bridge, audit, or end-to-end test is incomplete                             | A named delivery dependency whose completion is necessary before a production claim can be made.                                        |

### A.1 Minimum reproducibility record

A proving benchmark should publish enough information to distinguish protocol performance from harness effects. At minimum, record the circuit or statement identifier and gate count; proof-system and source revision; prover and verifier configuration; hardware, operating system, memory, and accelerator model; worker count and concurrency; cold-start and warm-resident behavior; sample count with median and tail latency; verification enabled or disabled; peak memory and, where material, energy or rental cost; and the stages excluded from the measurement. Throughput tests must identify whether jobs are independent proofs, recursive aggregation, or transactions represented by one proof. Any CPU/GPU comparison must use the same statement, proof format, verification path, and acceptance criteria.

## B Abbreviations and technical terms

**Table 8.** Abbreviations used in the report.

| Term       | Meaning in this report                                                                                                                                                 |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A2A        | Agent2Agent protocol: an interoperability interface for agent discovery, tasks, artifacts, and asynchronous state.                                                     |
| DA         | Data availability: the publication, retention, and recovery path required to reconstruct accepted state.                                                               |
| FHE        | Fully homomorphic encryption: computation over ciphertexts without revealing plaintext to the executing service.                                                       |
| IPA        | Inner-product argument: the polynomial-commitment proof mechanism used by Kimchi over the Pasta curves.                                                                |
| KMS / HSM  | Key-management system / hardware security module: enterprise infrastructure for controlled key custody and cryptographic operations.                                   |
| MCP        | Model Context Protocol: an interface through which agents connect to tools and contextual resources.                                                                   |
| MPC / SMPC | Multi-party computation / secure multi-party computation: protocols that distribute computation or control across parties without exposing each party’s private input. |
| MSM        | Multi-scalar multiplication: a dominant elliptic-curve commitment kernel and a principal GPU-acceleration target.                                                      |
| TEE        | Trusted execution environment: an attested hardware-isolated environment for confidential execution.                                                                   |
| zkApp      | A zero-knowledge application whose state transitions can require signatures or succinct proofs.                                                                        |
| zkEVM      | A proof system or rollup that establishes EVM-compatible execution and state transitions.                                                                              |
| zkVM       | A virtual machine that produces proofs of general program execution.                                                                                                   |
| ZKML       | Zero-knowledge machine learning: proofs or commitments for selected model, feature, provenance, or inference claims.                                                   |

## C Technology reference

This appendix expands the selected-stack summary in Section 1.4. It identifies the strongest role and material implementation boundary of each technology class so teams can select an edge component without confusing it with the proof-backed state and settlement substrate.

### Qualitative evaluation matrix

The numerical scores below are a heuristic design aid, not measurements or benchmark results. They summarize relative tendencies under the stated criteria and should be replaced by workload-specific evaluation before an implementation decision. Higher is better for the stated criterion; “generality” means ability to prove arbitrary execution, not suitability for the overall protocol.

**Table 9.** Qualitative technology evaluation matrix.

| Technology            | Primary job                            | Speed | Cost | Privacy | General | Settle. | Decision position                                                             |
|-----------------------|----------------------------------------|-------|------|---------|---------|---------|-------------------------------------------------------------------------------|
| A2A                   | Agent interoperability                 | 5     | 5    | 2       | 4       | 1       | Open edge compatibility; not a trust or settlement layer.                     |
| MCP                   | Tool interoperability                  | 5     | 5    | 2       | 4       | 1       | Tool compatibility behind mission-aware policy gates.                         |
| Signed capabilities   | Fast authority                         | 5     | 5    | 2       | 4       | 2       | Default low-risk fast path.                                                   |
| TEE / Phala CVM       | Confidential execution and attestation | 4     | 4    | 4       | 4       | 2       | Private low-latency execution; hardware trust remains.                        |
| Kimchi/o1js           | Application-specific ZK                | 2–4   | 3–5  | 5       | 2–3     | 4       | Private policy, state, and protocol claims.                                   |
| Pickles               | Recursive proof composition            | 2–4   | 3–5  | 5       | 3       | 5       | Compress mission receipts and checkpoints.                                    |
| RISC Zero / SP1       | General execution proof                | 2–4   | 2–4  | 5       | 5       | 3       | Arbitrary execution proof when needed.                                        |
| Zeko sovereign rollup | Private application state and batching | 4–5   | 4–5  | 5       | 4       | 5       | General private application-state substrate; mission state for agent domains. |
| Ethereum L1/EVM       | Global value and composability         | 1–3   | 1–3  | 2–3     | 4       | 5       | Assets, escrow, liquidity, and final settlement.                              |

### Technology roles and implementation boundaries

**Table 10.** Technology landscape for an agentic internet.

| Technology                                                | Principal property                     | Strongest role                                                                   | Material implementation boundary                                                                                |
|-----------------------------------------------------------|----------------------------------------|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Conventional encryption, TLS, and envelope key management | Confidentiality in transit and at rest | Default protection for mission data, artifacts, credentials, and storage objects | Plaintext handling during ordinary execution still requires a trusted runtime or confidential-compute boundary. |

| Technology                                                                            | Principal property                                                  | Strongest role                                                                                             | Material implementation boundary                                                                                               |
|---------------------------------------------------------------------------------------|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| Specialized privacy tools: PSI, searchable encryption, and attribute-based encryption | Private matching, encrypted retrieval, or policy-gated decryption   | Data-sharing workflows with a narrow, well-defined privacy operation                                       | Each addresses a specific operation rather than general confidential or verifiable computation.                                |
| Threshold cryptography and MPC[25]                                                    | Distribution of trust across independent parties                    | Key custody, threshold signatures, decryption, approval, and release policies                              | Interactive protocols and multi-party operation introduce latency and operational coordination.                                |
| TEEs and confidential VMs                                                             | Confidential execution with remote attestation                      | Low-latency private model and tool execution, confidential workers, and protected key release              | Assurance depends on hardware, firmware, host configuration, measured software, and key-management policy.                     |
| FHE[24]                                                                               | Computation directly on ciphertexts                                 | Selected computations or encrypted state that cannot be revealed to the executing service                  | Ciphertext operations and key management remain resource-intensive, so persistent encrypted computation must justify the cost. |
| Application-specific ZK, including Kimchi                                             | Succinct proof of a tailored predicate with private witness data    | Private policy, authorization, credentials, eligibility, provenance, solvency, and state-transition claims | Circuit design and proving costs rise with the generality of the statement.                                                    |
| ZKML                                                                                  | Proof or commitment of selected model, feature, or inference claims | High-assurance bounded inference and model-provenance cases                                                | Practicality depends on the model, architecture, and exact statement; it is not a general private-model-serving layer.         |
| zkVMs[26]                                                                             | General program-execution proofs                                    | Arbitrary software workflows, reproducible computation, and high-value execution claims                    | VM trace generation and proving are usually more expensive than a compact circuit for a recurring narrow claim.                |
| zkEVMs and ZK rollups[20]                                                             | Proof of EVM-compatible execution and batched state transitions     | EVM application compatibility, bridges, and Ethereum-oriented rollup settlement                            | EVM compatibility does not supply agent mission semantics or confidential execution by itself.                                 |
| STARKs[23]                                                                            | Transparent proof systems without a setup ceremony                  | Large computation and transparent-proof settings, including many zkVM and rollup designs                   | Prover cost, proof representation, recursion, and verifier integration remain workload-specific.                               |

| Technology                          | Principal property                                   | Strongest role                                                              | Material implementation boundary                                                                               |
|-------------------------------------|------------------------------------------------------|-----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| Recursive proofs, including Pickles | Composition of many proofs into a compact checkpoint | Mission trees, receipt streams, rollup batches, and settlement amortization | Recursive circuits have their own proving cost and are most useful when the aggregation workload justifies it. |

## References

- [1] Zeko Labs, “Technical Architecture,” <https://docs.zeko.io/architecture/technical-architecture>.
- [2] Zeko Labs, “Zeko Core Concepts,” <https://docs.zeko.io/architecture/zeko-core-concepts>.
- [3] Zeko Labs, “Security Model,” <https://docs.zeko.io/architecture/security-model>.
- [4] Zeko Labs, “Zeko Sequencer: Build Your Own Zeko Network,” <https://docs.zeko.io/operators/guides/sequencer>.
- [5] Zeko Labs, “Zeko’s Modular Data Availability Layer,” <https://docs.zeko.io/operators/guides/da-layer>.
- [6] Zeko Labs, “Launch a rollup on Phala,” <https://docs.zeko.io/operators/guides/rollup-on-phala>.
- [7] Zeko Labs, “Examples and Boilerplates,” including Agent Coordination Protocol, Agent Mission-Bound Auth, Verifiable AI Execution, and ZK x402 references, <https://docs.zeko.io/developers/examples-boilerplates>.
- [8] Zeko Labs, “SantaClawz Team Coordination Bridge V0.1,” private workshop roles, encrypted inter-agent envelopes, channels, receipts, and commitment-ledger architecture, [https://github.com/zeko-labs/santa\\_clawz-private\\_agents](https://github.com/zeko-labs/santa_clawz-private_agents).
- [9] Zeko Labs, “Agent Mission-Bound Auth,” protocol schemas and reference implementation for mission-scoped capabilities, holder-signed checkpoints, portable receipts, and Zeko registry anchors, <https://github.com/zeko-labs/agent-mission-bound-auth>.
- [10] Zeko Labs, “Ethereum ZK Rollup Framework: Technical Litepaper,” April 2026, <https://zeko.io/ethereumlitepaper.pdf>.
- [11] Succinct Prover Network Explorer, “Zeko-to-Ethereum batch settlement proof request 0x496fb75738253156a8d142653818a0b5b97a0aa85ceca1e4df7f52b937005e65,” fulfilled July 31, 2026 in Groth16 mode, <https://explorer.succinct.xyz/request/0x496fb75738253156a8d142653818a0b5b97a0aa85ceca1e4df7f52b937005e65>.
- [12] CoinMarketCap, “Succinct price today, PROVE to USD,” contemporaneous PROVE/USD reference, <https://coinmarketcap.com/currencies/succinct/>.
- [13] Zeko Labs, “x402 Zeko,” Ethereum and Base EVM payment rails, proof-gated reserve-release contracts, and Sepolia deployment and smoke paths, <https://github.com/zeko-labs/x402-zeko>.
- [14] o1Labs, “o1js Documentation,” <https://docs.o1labs.org/o1js>.
- [15] A2A Project, “Agent2Agent Protocol Specification,” release 1.0.0, <https://github.com/a2aproject/A2A/blob/main/docs/specification.md>.
- [16] Model Context Protocol, “Basic Specification,” <https://modelcontextprotocol.io/specification/2025-03-26/basic/index>.
- [17] Phala Cloud, “Attestation Overview,” <https://docs.phala.com/phala-cloud/attestation/overview>.
- [18] Phala Cloud, “Network Security,” <https://docs.phala.com/phala-cloud/networking/security>.
- [19] Ethereum Improvement Proposal 4844, “Shard Blob Transactions,” <https://eips.ethereum.org/EIPS/eip-4844>.
- [20] Ethereum Foundation, “Zero-knowledge rollups,” <https://ethereum.org/developers/docs/scaling/zk-rollups/>.
- [21] Ethereum Improvement Proposal 7841, “Cross-chain Message Format and Mailbox,” draft proposal for a neutral cross-chain messaging interface, <https://eips.ethereum.org/EIPS/eip-7841>.

- [22] Ethereum Foundation, “Bridges,” interoperability models, integration tradeoffs, and bridge risks, <https://ethereum.org/developers/docs/bridges/>.
- [23] StarkWare, “STARK Technology,” <https://starkware.co/stark/>.
- [24] Zama, “FHE on blockchain,” [https://docs.zama.ai/fhevm/fundamentals/architecture\\_overview/fhe-on-blockchain](https://docs.zama.ai/fhevm/fundamentals/architecture_overview/fhe-on-blockchain).
- [25] National Institute of Standards and Technology, “Multi-Party Threshold Cryptography,” <https://csrc.nist.gov/Projects/threshold-cryptography>.
- [26] RISC Zero, “Introduction to the RISC Zero zkVM,” <https://dev.risczero.com/>.
- [27] Arka Pal, Louai Zahran, William Gvozdzjak, Akilesh Potti, and Micah Goldblum, “Privacy-Preserving Mechanisms Enable Cheap Verifiable Inference of LLMs,” *arXiv:2602.17223*, February 2026, <https://arxiv.org/abs/2602.17223>.
- [28] Local workspace, “Same-statement proving acceleration,” companion file `docs/proving-acceleration.md`; see also `docs/proving-benchmarks.md` and `docs/native-agent-batching.md`.