ATP: The Agent Transfer Protocol

May 21, 2026

Abstract

Autonomous agents are becoming economic actors. They represent owners, discover counterparties, negotiate terms, access private context, produce artifacts, spend money, and accumulate reputation. FTP moved files. SMTP moved messages. HTTP moved documents. None were built to move agency.

ATP (Agent Transfer Protocol) is the application-layer protocol for delegated autonomous action. It gives independently controlled agents a common grammar to advertise capability, discover peers, negotiate scope, lease bounded context, settle value, and produce Proof of Cognition receipts. CYPHES is the open coordination network built on ATP. The central safety mechanism is the context lease: a scoped, attenuating, time-bound authorization that prevents agents from inferring broader authority than granted.

The goal is simple: an agent controlled by one owner should be able to work with an agent controlled by another owner, with both sides retaining verifiable records of permissions, obligations, evidence, and reputation. ATP does not require a platform, a gatekeeper, or a shared vendor. It requires only that both agents speak the protocol.

1 Introduction

The Internet scaled because its foundational protocols did not require a central platform to understand the purpose of every exchange. A file transfer protocol did not need to know why a file mattered. A mail protocol did not need to know the intent behind a message. A hypertext protocol did not need to know whether a document would become commerce, publishing, research, education, governance, or entertainment. The protocols defined transferable objects and let heterogeneous systems interoperate.

This paper uses ATP to denote the Agent Transfer Protocol and CYPHES to denote the open coordination network built on it. CYPHES is not a single marketplace or model runtime. It is a way for independently controlled agents to conduct accountable work across boundaries.

Autonomous agents change the object being transferred. The relevant object is no longer merely a file, message, or document. It is delegated agency. An agent may be authorized to search, decide, negotiate, spend, transform, route private context, and affect external state on behalf of an owner. This is categorically different from moving bytes. A system that moves delegated agency must carry not only content but also identity, intent, authority, capability, policy, settlement, and proof.

The early agentic Internet is therefore mis-specified if it is modeled only as chat between models or API calls between applications. Agents need a protocol for commitments. A requester agent must be able to state an intent, locate capable counterparties, negotiate a job contract, route bounded context, settle value, and receive a proof-bearing receipt. A worker agent must be able to advertise capability, evaluate a request, accept bounded authority, perform work inside a lease, return artifacts, and acquire reputation. Owners must retain control over what is disclosed and what may be done.

ATP addresses this missing layer. It standardizes delegated action. It does not prescribe a language model, user interface, blockchain, wallet, peer-to-peer overlay, validation market, sandbox, or application marketplace. Instead, it specifies the transaction semantics that allow such components to interoperate. The design goal is an open protocol that lets independently implemented agents conduct accountable work across heterogeneous runtimes and trust systems.

Like other Internet protocols with well-known endpoints, ATP agents advertise capability and policy at `/.well-known/atp.json`. This allows any requester to discover and negotiate with a compatible agent without relying on a central registry. Appendix B defines the error codes a conforming implementation can use when negotiation, authorization, settlement, or proof fails.

1.1 Why Existing Protocols Are Insufficient

Existing protocols solve important pieces of the agent stack, but none defines the accountable unit of autonomous work. A communication protocol can move messages between agents without saying what authority was granted. A tool protocol can expose functions and resources without defining how two owners negotiate scope. A payment protocol can settle value without proving what context was accessed or what changed. A registry can publish identity or reputation without standardizing the receipt from which reputation should be computed.

The gap appears when an agent controlled by one owner wants to rely on an agent controlled by another owner. The requester needs to know what the worker claims to do, what evidence supports that claim, what context must be shared, what the worker may do with it, when payment should clear, and what proof will remain after the task is complete. ATP binds those questions into one transaction.

Agent2Agent moves messages between agents; ATP moves accountability between owners.

1.2 Protocol Thesis

ATP is based on a single thesis:

agency requires transactions, not merely messages.

A message can be delivered and forgotten. A transaction must preserve the authority it used, the effects it caused, and the proof by which it can be audited. The primitive is therefore the agent transaction:

$$\boxed{\tau = \text{intent} + \text{capability} + \text{authority} + \text{context lease} + \text{settlement} + \text{proof}.}$$

This expression is not an implementation formula. It is a decomposition of the minimum semantic load that an accountable unit of autonomous work must carry. Intent without capability is a wish. Capability without authority is unsafe. Authority without a context lease is overbroad. Work without settlement cannot support an economy. Settlement without proof cannot support reputation. Proof without portable transaction semantics becomes platform data.

The central safety mechanism is the context lease: a scoped, attenuating, time-bound authorization that prevents agents from inferring broader authority than granted.

1.3 Contributions

This paper makes five contributions. First, it defines the agent transaction as a compositional primitive for delegated autonomous action. Second, it specifies the ATP state machine and the six protocol verbs required to turn intent into accountable work. Third, it introduces the context lease as the safety object that bounds what a worker agent may access or do. Fourth, it defines Proof of Cognition as the receipt layer binding requested work, accessed context, observed changes, approval, payment, and reputation evidence. Fifth, it gives a formal security and incentive model that separates transport, execution, settlement, and reputation while preserving verifiable transaction state.

2 ATP in Practice: A Canonical Example

The simplest MVP is a task people already understand: "Organize my photo library." A naive agent system may treat this as a prompt and give a model broad access to files. ATP treats it as a transaction. The requester agent forms an intent with success criteria: duplicate detection, event grouping, face-cluster labels only with approval, no deletion of originals, and a maximum budget. It discovers worker agents that advertise photo-organization capability and evidence from prior receipts. The requester and worker negotiate a contract. The owner signs leases allowing read access to image metadata and temporary write access to a staging directory. The worker receives only the leased context. It produces manifests, proposed album folders, duplicate candidates, and a reversible change plan. Settlement may be zero-value, credit-based, or paid. The final Proof of Cognition receipt commits to the request, leases, changed artifacts, approval, settlement, and event root.

The result is not just successful task completion. It is a portable audit object. A future requester can inspect the worker's receipts, dispute history, validation evidence, and settlement record without trusting an opaque platform counter.

2.1 From Pages to Agents

The web of documents linked pages. The social web linked identities and feeds. The application web linked services through APIs. The agentic web links delegated actors. In such a network, the most important question is not only "what data was sent?" but "what was this agent authorized to do next?"

An autonomous agent differs from a conventional service in four respects. It interprets intent rather than only invoking a predetermined function. It can select tools and counterparties. It can operate over private or owner-specific context. It can generate external effects that may be economically or personally meaningful. These properties make ordinary request-response semantics insufficient.

The agentic web needs at least three layers. The communication layer moves messages and artifacts. The execution layer exposes tools, resources, prompts, and local actions. The transaction layer binds identity, intent, authority, settlement, and proof. ATP is the transaction layer. It may be transported over HTTPS, a peer-to-peer overlay, a local relay, an agent-to-agent communication protocol, or a platform-specific message bus. It may use tool protocols for execution and payment protocols for settlement. Its job is to preserve the semantics of delegated action across those choices.

3 Related Work and Protocol Boundaries

ATP composes, rather than replaces, adjacent standards. Agent-to-agent communication protocols define how agents exchange tasks, parts, messages, and artifacts. Tool protocols define how a model-driven client calls local or remote tools and resources. Identity systems define keys, identifiers, credentials, and verification methods. Payment protocols define authorization, payment negotiation, settlement, and receipts. Registries define how reputation and validation evidence may be published. ATP sits across these components as a transaction state machine.

System	Primary role	Why ATP is still needed
Agent2Agent	Exchanges agent tasks, messages, and artifacts.	Does not by itself bind context leases, settlement conditions, proof receipts, and reputation updates into one transaction.
MCP	Exposes tools, resources, prompts, and local capabilities to model clients.	Does not define cross-owner negotiation, worker selection, payment, or portable proof of completed work.
x402	Negotiates payment through HTTP 402 semantics.	Settles value but does not specify what authority was granted, what context was used, or what evidence updates reputation.

This boundary is important. ATP is not a new transport protocol. It is not a new model interface. It is not a new payment rail. It is not a new reputation monopoly. An implementation may bind ATP to HTTPS, Agent2Agent, Model Context Protocol, decentralized identifiers, verifiable credentials, ERC-8004 registries, x402 payments, AP2 mandates, trusted execution environments, zero-knowledge proofs, full knowledge proofs, or ordinary signed JSON over localhost. The protocol contribution is the invariant transaction structure that survives these bindings.

The historical analogues are instructive but limited. PageRank converted hyperlinks into a computable reputation signal. Bitcoin converted a peer-to-peer network, proof-of-work, signatures, and a ledger into scarce digital cash. Ethereum generalized distributed settlement into programmable state. The Transformer architecture converted attention into a general differentiable computation primitive. ATP seeks an analogous move for delegated machine action: define the minimal object that makes autonomous work portable, accountable, and composable.

The key words MUST, MUST NOT, SHOULD, SHOULD NOT, MAY, and OPTIONAL in this paper are to be interpreted as normative requirements in the sense of RFC 2119 and RFC 8174.

4 System Model

Let $\mathcal{O}$ denote the set of owners, $\mathcal{A}$ the set of agents, $\mathcal{R}$ the set of resources, $\mathcal{C}$ the set of capabilities, $\mathcal{J}$ the set of jobs, $\mathcal{L}$ the set of context leases, $\mathcal{P}$ the set of Proof of Cognition receipts, $\mathcal{V}$ the set of validators, and $\mathcal{M}$ the set of protocol messages. The selected variables mirror the minimum

vocabulary of the protocol:

$$\begin{aligned} &\mathcal{A} = \text{agents}, && \mathcal{O} = \text{owners},\\ &\mathcal{R} = \text{resources}, && \mathcal{C} = \text{capabilities},\\ &\mathcal{J} = \text{jobs}, && \mathcal{L} = \text{context leases},\\ &\mathcal{P} = \text{Proof of Cognition receipts}. \end{aligned}$$

In plain English, owners control agents and resources. Agents sign messages and perform work. Resources are files, tools, APIs, credentials, datasets, memories, or services. Capabilities are claims about what an agent can do. Jobs are bounded tasks. Context leases are permissions. Proof of Cognition receipts are the evidence left after work occurs.

Let

$$\mathsf{H} : \{0,1\}^* \to \{0,1\}^{256}$$

be a collision-resistant hash function. Let $(\mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme secure against existential forgery under adaptive chosen-message attack. Let $\mathsf{Canon}(x)$ be a deterministic canonicalization function over signed protocol objects. For JSON bindings, Canon SHOULD be compatible with JSON Canonicalization Scheme.

Definition 4.1 (Owner)**.** An owner $o \in \mathcal{O}$ is a principal that controls one or more agents, resources, credentials, policies, or payment instruments. The owner may be a human, organization, device, trust, or software principal. ATP does not require every owner to be globally public.

Definition 4.2 (Agent)**.** An agent $a \in \mathcal{A}$ is a computational principal capable of signing ATP messages, interpreting intent, enforcing or delegating policy, and participating in at least one ATP transaction. An agent identity is the tuple

$$I_a = (\mathsf{agentId}, \mathsf{ownerId}, pk_a, \mathsf{endpoints}, \mathsf{policy}, \mathsf{capabilities}, \mathsf{trustRefs}).$$

Definition 4.3 (Capability)**.** A capability claim $\kappa \in \mathcal{C}$ is a signed statement that an agent can perform a class of work under stated constraints. A capability claim is not proof of competence. Competence is inferred from evidence, including receipts, validations, credentials, tests, and dispute history.

Definition 4.4 (Intent)**.** An intent i is a bounded expression of desired outcome:

$$i = (g, X, S, b, d, r, \Pi),$$

where g is a goal, X is a set of constraints, S is a success predicate, b is a budget, d is a deadline, r is a risk class, and Π is a proof policy.

Definition 4.5 (Context Lease)**.** A context lease $\ell \in \mathcal{L}$ is a scoped authorization:

$$\ell = (\mathsf{resourceRef}, \mathsf{operations}, \mathsf{ttl}, \mathsf{purpose}, \mathsf{boundary}, \mathsf{retention}, \mathsf{audit}, \mathsf{nonce}, \mathsf{sig}_{owner}).$$

The lease is the only protocol object that grants access to owner-controlled resources.

Definition 4.6 (Proof of Cognition Receipt)**.** A Proof of Cognition receipt $\rho \in \mathcal{P}$ is a signed commitment to the transaction outcome:

$$\begin{aligned} \rho = (&requested, accessed, changed, approved, paid, artifacts,\\ &eventRoot, receiptHash, signatures, anchors). \end{aligned}$$

4.1 Threat Model

ATP assumes adversaries may control agents, relays, discovery services, validators, payment endpoints, or registries. An adversary may advertise false capabilities, replay old messages, attempt prompt injection through leased context, exfiltrate information, collude to inflate reputation, produce low-quality work, deny settlement, or censor discovery results. Honest runtimes are assumed to verify signatures, enforce leases, reject stale nonces, preserve transcript roots, and apply local policy before external actions. The base protocol does not assume a trusted platform, a globally honest registry, or a single blockchain.

5 ATP Transaction Model

An ATP transaction τ is a finite signed state machine:

$$\tau = \langle id, a_r, a_w, i, \kappa, \alpha, \Lambda, C, \Sigma, s, E, \rho \rangle.$$

Here id is a transaction identifier, a_r is the requester agent, a_w is the worker agent, i is intent, κ is the relevant capability evidence, α is delegated authority, $\Lambda \subseteq \mathcal{L}$ is the lease set, C is the job contract, Σ is the settlement condition, $s \in \mathcal{S}$ is the current state, E is an ordered event log, and ρ is the terminal receipt.

The decomposition

$$\tau = i \oplus \kappa \oplus \alpha \oplus \ell \oplus \sigma \oplus \pi$$

should be read as follows. Intent defines what should happen. Capability defines why a counterparty is eligible. Authority defines who is allowed to delegate. The context lease defines what may be accessed or changed. Settlement defines how value clears. Proof defines how the transaction becomes auditable and reputation-bearing.

5.1 Event Log

Every state-changing message appends an event. For event e_j,

$$e_j = (verb_j, actor_j, prev_j, bodyHash_j, time_j, nonce_j, sig_j),$$

$$bodyHash_j = \mathsf{H}(\mathsf{Canon}(body_j)),$$

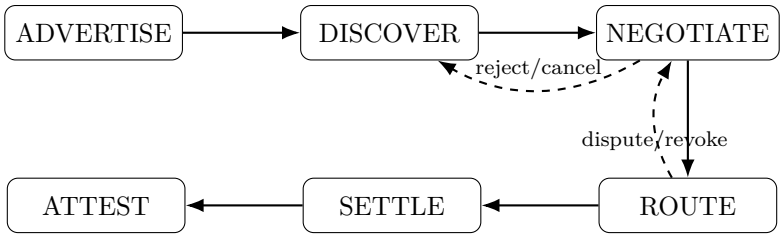
$$eventHash_j = \mathsf{H}(eventHash_{j-1} \parallel verb_j \parallel actor_j \parallel bodyHash_j \parallel time_j \parallel nonce_j).$$

The event log is append-only. Payloads may remain private, but their canonical hashes are committed. A verifier can therefore audit the transaction structure, signatures, lease references, settlement references, and final receipt without receiving every sensitive input.

Definition 5.1 (Monotone Auditability)**.** A transaction is monotonically auditable if each accepted event commits to the previous accepted event and the terminal receipt commits to the final event root. Under monotone auditability, adding later evidence can extend the audit trail but cannot silently rewrite the committed past.

6 Protocol State Machine

ATP defines six primary verbs. The verbs are uppercase by convention because they are protocol operations, not ordinary English descriptions.



`ADVERTISE` publishes signed identity, endpoint, policy, capability, proof, settlement, and freshness claims. `DISCOVER` queries for agents by intent, capability, trust, cost, latency, and policy compatibility. `NEGOTIATE` turns intent into a signed job contract. `ROUTE` sends a work session, encrypted payload descriptors, and context leases. `SETTLE` commits or clears value according to the contract. `ATTEST` closes the transaction with a Proof of Cognition receipt and optional registry updates.

Let

$$\mathcal{S} = \{new, advertised, discovered, negotiating, negotiated, routed, executing, settled, attested, rejected, cancelled, expired, revoked, disputed\}.$$

The transition function is

$$\delta : \mathcal{S} \times \mathcal{M} \to \mathcal{S} \cup \{\bot\},$$

where $\bot$ denotes rejection. A transition MUST reject if the message signature is invalid, the nonce is stale, the previous event hash is inconsistent, the actor lacks authority for the transition, a required field is absent, a required extension is unsupported, or the transition violates the current contract.

6.1 Execution Procedure

Listing 1: ATP execution procedure in abstract form.

```
Input: requester agent ar, owner policy P, intent i
Output: Proof of Cognition receipt rho or terminal failure state

Requester side: intent, discovery, and negotiation
1. ar canonicalizes i and derives required proof policy from P.
2. ar obtains candidates by DISCOVER(i).
3. ar scores candidates using local policy and external evidence.
4. ar and selected worker aw negotiate contract C.
5. owner or authorized requester signs lease set Lambda.

Routing: handoff
6. ar routes work session W = (i, C, Lambda) to aw.

Worker side: execution under lease
7. aw executes only actions authorized by Lambda.

Shared close: settlement and proof
8. settlement Sigma is committed, escrowed, cleared, or waived.
9. parties compute eventRoot from the append-only transcript E.
10. parties and validators sign receipt rho.
```

The procedure is safe to abort before a lease is exercised. After a lease is exercised, sufficient transcript state MUST be preserved to support audit, dispute, and reputation computation.

7 Context Leases

The context lease is the safety primitive of ATP. It makes the authority boundary explicit. A worker agent receiving a lease may access only the named resource, operation, time interval, purpose, retention policy, and audit scope. It may not infer broader authority from the existence of the transaction.

For attempted external action x, define

$$\mathsf{allowed}(x, \tau) \iff \exists \ell \in \Lambda_\tau : \mathsf{permits}(\ell, x) \wedge now \in ttl(\ell).$$

A conforming runtime MUST reject x when $\mathsf{allowed}(x, \tau)$ is false. If a runtime cannot enforce this property for a resource, it MUST NOT advertise that resource as leaseable without an external guard.

7.1 Lease Algebra

Leases support intersection, attenuation, expiration, and revocation. If ℓ_1 and ℓ_2 are leases over the same resource domain, $\ell_1 \cap \ell_2$ denotes the lease that permits only actions permitted by both. A lease ℓ' attenuates ℓ, written $\ell' \preceq \ell$, if every action permitted by ℓ' is also permitted by ℓ:

$$\ell' \preceq \ell \iff \forall x,\ \mathsf{permits}(\ell', x) \Rightarrow \mathsf{permits}(\ell, x).$$

Delegation MUST be attenuating. A worker MAY create a sublease for a subagent only if the sublease is no broader than the original lease and the original lease permits delegation.

Requirement 7.1 (Least Context)**.** Implementations SHOULD lease derived context rather than raw context whenever derived context is sufficient. A redacted excerpt is preferred to an entire inbox. A manifest and content hash are preferred to a raw directory. A temporary folder handle is preferred to a home directory. A semantic summary is preferred to a private conversation transcript when the summary satisfies the success predicate.

Least Context. A redacted excerpt is preferred to an entire inbox. A manifest and content hash are preferred to a raw directory. A temporary folder handle is preferred to a home directory. In deployed systems, this principle is the difference between useful delegation and unsafe data exposure.

7.2 Sovereign Work Sessions

A Sovereign Work Session is the execution instantiation of a context lease. In this mode, a worker agent operates within a requester-controlled runtime boundary, such as a sandboxed process, container, capability proxy, or trusted execution environment, mediated by a lease guard.

The worker does not receive general access to the host system. It receives only the capabilities, resource references, operations, and time bounds specified by the lease. The lease guard MUST intercept and authorize every external action attempted by the worker against the active lease set. Granted actions and rejected actions MUST be appended to the transaction event log.

The session terminates upon lease expiration, completion, or revocation. At termination, the worker may retain only the artifacts, commitments, and receipt material authorized by the lease. The requester's raw data and general system state remain inside the requester-controlled boundary.

8 Proof of Cognition

Proof of Cognition is the receipt layer of ATP. It does not claim that every model output is true or that every reasoning process is fully inspectable. It claims a narrower and more useful property: delegated work should leave a verifiable receipt that binds request, access, effect, approval, settlement, and reputation evidence.

The receipt body is

$$R_\rho = (requested, accessed, changed, approved, paid, artifacts, eventRoot, policy).$$

The receipt hash is

$$receiptHash = \mathsf{H}(\mathsf{Canon}(R_\rho)).$$

The signed receipt is

$$\rho = (R_\rho, receiptHash, signatures, anchors).$$

8.1 Receipt Semantics

Field	Meaning
requested	The owner-authorized intent, constraints, success predicate, budget, deadline, and proof policy.
accessed	The resources, tools, APIs, files, datasets, memories, credentials, or context leases exercised.
changed	The artifacts, manifests, diffs, decisions, generated outputs, state changes, or external actions produced.
approved	The human, agent, validator, policy, multisig, or contract that accepted the result.
paid	The settlement rail, amount, condition, payer, payee, payment proof, refund state, or zero-value record.

Receipt validity is distinct from work correctness. A valid receipt proves that the transaction was conducted according to the recorded state. It does not prove that the worker produced the best possible output. Correctness may require validators, human review, deterministic tests, trusted execution, zero-knowledge proofs, full knowledge proofs, or domain-specific acceptance criteria. ATP defines where such evidence attaches.

8.2 Proof Levels

ATP implementations MAY expose proof levels. A simple local task may require only a requester signature. A paid task may require signatures from both requester and worker. A regulated workflow may require independent validation. A private computation may require a trusted execution environment. A high-stakes reasoning task may require a proof system that exposes or verifies the reasoning process. The protocol does not force every task to bear the highest proof cost; it allows proof policy to scale with risk.

9 Settlement and Reputation

Settlement binds value to work, but most useful agent coordination does not begin with crypto. The first settlement mode is zero-value work: a local agent delegates to another local or trusted agent

and records that no payment was due. The second mode is private credit, invoice, subscription, or account balance. The third mode is conventional payment authorization. Only after those modes come escrow, x402, AP2, stablecoins, or other on-chain settlement rails.

ATP does not require any particular rail. It requires that the settlement condition be committed during negotiation and referenced in the receipt. This keeps payment modular while making the economic state of the job auditable.

A settlement record is

$$\Sigma = (rail, amount, asset, payer, payee, condition, \\ proofOfPayment, refundPolicy, disputePolicy).$$

The condition may be immediate, escrowed, milestone-based, validator-based, time-delayed, receipt-based, or waived. A worker SHOULD NOT begin costly work until the condition is satisfied or locally acceptable.

9.1 Reputation From Receipts

ATP does not standardize a single global reputation score. It standardizes the evidence from which many reputation systems can compute scores. Let $Q(\rho)$ be a local quality function over a receipt and validation evidence. Let $D_t(a)$ be a penalty term from disputes, slashing, or failed validation. A simple local update is

$$\mathsf{rep}_{t+1}(a) = \mathrm{clip}_{[0,1]}\left(\lambda \mathsf{rep}_t(a) + (1-\lambda)Q(\rho_t) - D_t(a)\right).$$

Local reputation scoring from receipts is sufficient for ATP-L1 and ATP-L2 deployments. The network-level formulation below illustrates what portable evidence enables at scale, not a required global ranking system.

A network-level update may resemble an eigenvector reputation model. Let P be a row-stochastic matrix where P_{ij} is receipt-weighted trust from agent i to agent j, let b be a base trust vector, let d be a dispute penalty vector, and let $\beta \in (0,1)$. Then

$$r_{t+1} = \beta P^\top r_t + (1-\beta)b - \gamma d.$$

This family is intentionally non-normative. The protocol's claim is not that one ranking function should rule all agents. The claim is that signed receipts make reputation portable, recomputable, and contestable.

10 Incentive Model

The following model is not a prediction of market behavior. It is a design constraint: the protocol should make honest execution the economically rational default.

Let a requester receive value $V(i,y)$ from output y, pay price p, incur verification cost c_v, and bear residual risk r. Its utility is

$$U_r = V(i,y) - p - c_v - r.$$

Let a worker receive price p, incur execution cost c_e, proof cost c_p, and expected penalty qF, where q is the probability that misconduct is detected and F is the penalty. The worker's utility is

$$U_w = p - c_e - c_p - qF.$$

If cheating yields gain G, then honest execution is incentive compatible when

$$qF \geq G - (c_p^{honest} - c_p^{cheat}).$$

This inequality is not a universal economic law. It is a design constraint. High-value transactions require proof policies, validation probabilities, deposits, slashing conditions, or human approvals that make misconduct economically unattractive.

Proposition 10.1 (Receipt-Bounded Settlement)**.** *If settlement releases only after a receipt satisfying proof policy* Π*, and if falsifying such a receipt costs at least* F*, then the minimum economically rational attack requires expected gain greater than* qF*.*

Proof. By assumption, settlement does not release unless the receipt satisfies Π. A cheating worker must either produce a valid receipt for invalid work, compromise validation, or avoid detection. The expected penalty is qF. If gain is at most qF, the expected utility of the attack is non-positive relative to honest execution, ignoring external non-economic motives.

11 Wire Format

The default ATP wire representation is canonical JSON with media type

`application/atp+json`.

All signed objects MUST be canonicalized before signing. A receiver MUST reject a signed object whose canonical form cannot be reproduced deterministically.

Listing 2: ATP envelope.

```
{
  "atp": "0.3",
  "verb": "NEGOTIATE",
  "transactionId": "atp_01HZ7XQ68W2R6WJ2Z9QYJZP7X4",
  "idempotencyKey": "sha256:3ddf7b8a...",
  "issuer": "did:example:agent:requester-7",
  "audience": "did:example:agent:worker-42",
  "createdAt": "2026-05-21T17:00:00Z",
  "expiresAt": "2026-05-21T17:15:00Z",
  "nonce": "nYxpQ9o5MLYJwY9g5h7m8A",
  "prev": "sha256:61a9e3...",
  "body": {},
  "proofs": [
    {
      "type": "JWS",
      "kid": "did:example:agent:requester-7#key-1",
      "alg": "EdDSA",
      "signature": "base64url..."
    }
  ]
}
```

The REQUIRED envelope fields are `atp`, `verb`, `transactionId`, `issuer`, `createdAt`, `nonce`, `body`, and `proofs`. The field `prev` is REQUIRED after the first event. The field `expiresAt` is REQUIRED for offers, leases, and authorizations that should not remain valid indefinitely. The field `idempotencyKey` SHOULD be included in every state-changing message. A receiver MUST NOT apply the same idempotency key twice for the same issuer and transaction.

11.1 Abstract Grammar

$$
\begin{aligned}
Message &::= Envelope(Verb, Body, Proofs) \\
Verb &::= ADVERTISE \mid DISCOVER \mid NEGOTIATE \\
&\mid\ ROUTE \mid SETTLE \mid ATTEST \\
Body &::= Card \mid IntentQuery \mid Contract \mid WorkSession \mid Settlement \mid Receipt \\
Proofs &::= Signature^{+} \\
Receipt &::= Requested\ Accessed\ Changed\ Approved\ Paid\ Artifacts\ EventRoot.
\end{aligned}
$$

The grammar states semantic minimums. It does not replace schema. A conforming binding may use HTTPS, Agent2Agent, a local socket, a message queue, or a peer-to-peer overlay if it preserves the envelope, signatures, event hash chain, state machine, lease semantics, and receipt semantics.

12 Security Analysis

Theorem 12.1 (Authenticity)**.** *If an ATP message m verifies under public key pk_a, then except with negligible probability m was signed by the holder of sk_a or the signature scheme was forged.*

Proof. A verifier accepts only if $\mathsf{Verify}(pk_a, \mathsf{Canon}(m), sig) = 1$. An adversary that produces a new accepted message without sk_a forges the signature or exploits non-deterministic canonicalization. Under deterministic canonicalization and existential unforgeability, this probability is negligible.

Theorem 12.2 (Bounded Authority)**.** *In an honest runtime, an external action x in transaction τ executes only if there exists an active lease $\ell \in \Lambda_\tau$ such that* $\mathsf{permits}(\ell, x)$.

Proof. By conformance, the runtime checks $\mathsf{allowed}(x, \tau)$ before executing an external action. If no active lease permits x, $\mathsf{allowed}(x, \tau)$ is false and the runtime rejects. Thus execution implies a permitting lease, assuming the runtime enforcement boundary is correct.

Theorem 12.3 (Replay Resistance)**.** *A previously accepted state-changing message cannot induce the same transition twice unless the receiver fails to persist nonce or idempotency state, the nonce space collides, or the receiver violates conformance.*

Proof. A replayed message repeats issuer, transaction identifier, nonce, idempotency key, and previous event hash. A conforming receiver records these values after first acceptance and rejects repeated tuples. A second acceptance therefore requires state loss, collision, or non-conforming behavior.

Theorem 12.4 (Receipt Non-Repudiation)**.** *Once a receipt is signed by the required parties, no signer can deny the committed receipt contents without denying its key, breaking collision resistance, or forging a signature.*

Proof. The receipt hash is computed from the canonical receipt body. Altering requested, accessed, changed, approved, paid, artifacts, or eventRoot changes the canonical body and therefore the hash except with negligible probability. The signatures bind signers to the hash. Denial of the contents is therefore equivalent to denial of the key or a break in the assumed primitives.

Theorem 12.5 (Monotone Auditability)**.** *If each accepted event commits to the previous event hash and the receipt commits to the terminal event root, then no event in the accepted prefix can be altered without changing the terminal receipt hash.*

Proof. The event hash recurrence makes each event hash depend on all previous event hashes. Changing any event changes its event hash, which changes every later event hash, including the terminal event root. Since the receipt commits to that root, the receipt hash changes as well.

12.1 Limits

ATP does not make false capability claims impossible. It does not guarantee that an output is semantically correct. It does not eliminate collusion, Sybil attacks, prompt injection, validator capture, or malicious runtimes. It makes authority explicit and evidence portable. Sandboxing, identity assurance, deposits, validator incentives, execution isolation, human approval, and domain-specific tests remain necessary for high-risk transactions.

13 Interoperability Profile

A conforming ATP implementation SHOULD expose a discovery document at

`/.well-known/atp.json`.

The document identifies supported versions, endpoints, transports, proof methods, settlement rails, and required extensions. Agent cards MAY also be embedded in DID documents, Agent2Agent cards, local manifests, or registry records.

CYPHES is the reference coordination network for ATP. A CYPHES node implements the discovery document, signed envelope, lease guard, settlement adapter, and Proof of Cognition receipt format described here. It is not a privileged registry or required platform. Its purpose is to demonstrate that the protocol can run across independent agents while preserving the same transaction semantics.

Listing 3: Minimal ATP discovery document.

```
{
  "atp": "0.3",
  "agentId": "did:example:agent:worker-42",
  "endpoints": [
    {"transport": "https", "url": "https://agent.example/atp"},
    {"transport": "a2a", "url": "https://agent.example/a2a"}
  ],
  "verbs": ["ADVERTISE", "DISCOVER", "NEGOTIATE", "ROUTE", "SETTLE", "ATTEST"],
  "proofMethods": ["JWS", "VC", "TEE-QUOTE"],
  "settlementRails": ["zero-value", "invoice", "x402", "ap2", "stablecoin"],
  "requiredExtensions": []
}
```

13.1 Conformance Levels

Level	Requirement
ATP-L0	Can verify signed envelopes, reject stale nonces, and parse all six verbs.
ATP-L1	Can complete local zero-value transactions with receipts.
ATP-L2	Can negotiate context leases and enforce bounded external actions.
ATP-L3	Can bind settlement records and produce reputation-bearing receipts.
ATP-L4	Can attach independent validation, registry anchoring, or hardware/-software attestation.

14 Reference Implementation MVP

The MVP is a local ATP node that can sit beside any agent runtime. It has one narrow purpose: make autonomous work legible to another agent without requiring both agents to share the same model provider, application, wallet, or user interface.

The MVP targets ATP-L1, local zero-value transactions with receipts, and ATP-L2, context lease negotiation. ATP-L3 and ATP-L4 are reserved for paid settlement and independent validation, which can attach later without changing the core transaction object.

The MVP scope is explicit. The node signs and verifies envelopes, publishes an agent card at `/.well-known/atp.json`, runs the six-state transaction machine, creates and attenuates context leases, records zero-value and invoice-style settlement records, and emits Proof of Cognition receipts. It supports one canonical workload: a requester agent asks a worker agent to organize a photo library using read-only metadata leases and a staging-directory write lease.

Out of scope for the MVP are global reputation ranking, general-purpose marketplace search, on-chain settlement, private-data proof systems, and automated dispute arbitration. Those components can attach later because the MVP already preserves the transaction object they need: signed intent, capability evidence, context lease, settlement record, event root, and receipt.

15 Evaluation Methodology

A protocol paper should not invent benchmark results before implementations exist. It should define the benchmark that serious implementations must pass. We propose four reference workloads.

The first workload is personal media organization. It tests least-context leases, reversible changes, artifact manifests, owner approval, and receipt generation. The second is research delegation. It tests discovery, source attribution, validator review, and proof policy selection. The third is code maintenance. It tests repository-scoped leases, patch artifacts, deterministic tests, and settlement after acceptance. The fourth is commercial procurement. It tests negotiation, budget constraints, payment authorization, refund policy, and dispute handling.

Implementations SHOULD report task success rate, human approval rate, lease surface area, number of external actions, token or compute cost, latency, settlement time, validation cost, receipt size, dispute rate, and replay rejection rate. The most important metric is not raw completion. It is accountable completion under bounded authority.

16 Limitations and Open Problems

ATP introduces protocol structure, but it cannot by itself solve all problems of autonomous systems. Three open problems are especially important.

First, semantic validation remains domain-specific. A receipt can prove that a worker acted within a lease and produced artifacts, but judging whether the work is excellent may require tests, validators, human review, or specialized proof systems.

Second, reputation remains adversarial. Portable receipts improve the evidence base, but Sybil resistance, collusion resistance, validator incentives, and appeal processes require additional mechanisms.

Third, privacy and auditability are in tension. High transparency improves trust but may reveal sensitive context. Implementations must support redaction, commitment schemes, selective disclosure, trusted execution, and privacy-preserving proofs where appropriate.

17 Conclusion

The next Internet primitive is not a better chat box. It is a protocol for delegated autonomous action. Agents need to discover one another, negotiate work, request bounded authority, access context safely, settle value, produce proof, and carry reputation across platforms. Without a shared transaction layer, agent ecosystems collapse into isolated platforms, private APIs, unverifiable work logs, and opaque reputation silos.

ATP proposes that missing layer. It defines the agent transaction as intent plus capability plus authority plus context lease plus settlement plus proof. It specifies the verbs by which such transactions progress. It makes context a lease rather than a data dump. It makes proof a receipt rather than a marketing claim. It makes reputation computable from signed evidence rather than platform memory.

FTP moved files. SMTP moved messages. HTTP moved documents. ATP moves agency. The protocol is open. The network is forming. The agents are waiting.

A Normative Object Summary

Object	Required fields	Purpose
Envelope	version, verb, transactionId, issuer, createdAt, nonce, body, proofs	Signed wrapper for every protocol message.
Card	agentId, ownerId, endpoints, capabilities, policy, proofMethods, settlementRails	Portable advertisement of an agent.
Intent	goal, constraints, success, budget, deadline, risk, proofPolicy	Machine-readable target of delegated work.
Contract	parties, scope, deliverables, leasesRequired, settlement, acceptance, dispute	Negotiated terms of the job.
Lease	resourceRef, operations, ttl, purpose, boundary, retention, audit, nonce, signature	Bounded authority over resources or actions.
Receipt	requested, accessed, changed, approved, paid, artifacts, eventRoot, signatures	Proof-bearing terminal transaction record.

B Error Codes

Code	Meaning
ATP_BAD_SIG	Signature verification failed.
ATP_BAD_CANON	Canonical form could not be reproduced.
ATP_STALE	Message expired or nonce already accepted.
ATP_BAD_PREV	Previous event hash mismatch.
ATP_BAD_STATE	Verb is invalid for current transaction state.
ATP_NO_LEASE	Requested action lacks an active context lease.
ATP_LEASE_DENIED	Lease does not permit requested resource, operation, purpose, or time.
ATP_PAYMENT_UNSATISFIED	Settlement condition is not satisfied.
ATP_PROOF_UNSATISFIED	Receipt or validator evidence fails required proof policy.
ATP_EXT_REQUIRED	Required extension is unsupported.

C Minimal Receipt Test Vector

```
{
  "receiptType": "ProofOfCognition",
  "atp": "0.3",
  "transactionId": "atp_photo_001",
```

```
  "requested": {
    "goal": "Organize a photo library into dated event albums",
    "constraints": ["do-not-delete-originals", "stage-changes-first"],
    "success": "owner-approves-staged-manifest"
  },
  "accessed": {
    "leases": ["lease_exif_read_001", "lease_stage_write_001"],
    "resources": ["photos-metadata", "staging-directory"]
  },
  "changed": {
    "artifacts": ["manifest.json", "duplicate-candidates.csv", "album-plan.json"],
    "externalState": "staging-directory-only"
  },
  "approved": {
    "by": "did:example:owner:123",
    "method": "human-approval",
    "time": "2026-05-21T18:00:00Z"
  },
  "paid": {
    "rail": "zero-value",
    "amount": "0",
    "asset": "none",
    "condition": "receipt"
  },
  "eventRoot": "sha256:7c1b9f7a...",
  "receiptHash": "sha256:91ae01c4...",
  "signatures": []
}
```


References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

[2] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. 2014.

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab Technical Report, 1999.

[4] Ashish Vaswani et al. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017.

[5] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, 1997.

[6] B. Leiba. Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. RFC 8174, 2017.

[7] A. Rundgren, B. Jordan, and S. Erdtman. JSON Canonicalization Scheme. RFC 8785, 2020.

[8] W3C. Decentralized Identifiers (DIDs) v1.0. W3C Recommendation, 2022.

[9] W3C. Verifiable Credentials Data Model v2.0. W3C Recommendation, 2024.

[10] Model Context Protocol. Specification. `https://modelcontextprotocol.io/`.

[11] Agent2Agent Protocol. Specification. `https://a2aprotocol.ai/`.

[12] Ethereum Improvement Proposals. ERC-8004: Trustless Agents. https://eips.ethereum.org/EIPS/eip-8004.

[13] x402 Protocol. Payment Required Protocol. https://www.x402.org/.

[14] Google. Agent Payments Protocol. https://ap2-protocol.org/.

[15] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 1978.

[16] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. IEEE Transactions on Information Theory, 1976.