theailab.org

STANDARD | VERSION 1.5

TIP for AI Agents

Verified Agent Identity, Delegated Authority and Verifiable Actions on the Trust Identity Protocol

AUTHOR
Dinesh Mendhe, Founder and Chairman
ORCID 0000-0003-1158-3259

PUBLISHER
The AI Lab Intelligence Unobscured, Inc.
Wilmington, Delaware, United States

DATE
November 19, 2026

DOI
10.5281/zenodo.24359932

LICENSE
Creative Commons Attribution 4.0 International

STATUS
Agent profile for deployment; ledger changes by RFC

theailab.org

Contents

Suggested Citation

Foreword

Acknowledgments

Executive Summary

Key terms

Part I. The Problem

1. The problem

2. Current issues, 2025 and 2026

3. Use cases

Part II. Landscape and Design Principles

4. What exists today, and the gap

5. Design goals and non-goals

Part III. Architecture and Agent Identity

6. Architecture overview

7. Agent identity

Part IV. Delegated Authority and Human Control

8. Mandates: delegated authority

9. TIP Approval for high-risk actions

Part V. Actions, Verification and Evidence

10. Action requests

11. Verification at the receiving site

12. Receipts, logs and anchoring

Part VI. Revocation, Safety and Agent-Made Content

13. Revocation and TIP Stop All

14. Agent-made content and Verifiable Assets

Part VII. Inputs, Accountability and Security Analysis

15. Protecting the agent's inputs

16. Accountability and reputation

17. Threat model

Part VIII. Interoperability, Deployment and Scale

theailab.org

18. Interoperability

19. Scalability and performance

Part IX. Privacy, Regulation and Economics

20. Privacy and data protection

21. Regulation and sector requirements

22. Economics and lookup costs

Part X. Adoption

23. Changes needed in TIP

24. Roadmap

25. What this design does not do

26. Open questions for The AI Lab and the AI Trust Council

Appendices

Appendix A. Example records

Appendix B. Verifier outline

Appendix C. How this paper builds on pull request 383

References

theailab.org

Suggested Citation

APA. Mendhe, D. (2026). *TIP for AI Agents: Verified agent identity, delegated authority and verifiable actions on the Trust Identity Protocol* (Version 1.5). The AI Lab Intelligence Unobscured, Inc. https://doi.org/10.5281/zenodo.24359932

Chicago. Mendhe, Dinesh. 2026. *TIP for AI Agents: Verified Agent Identity, Delegated Authority and Verifiable Actions on the Trust Identity Protocol*. Version 1.5. Wilmington, DE: The AI Lab Intelligence Unobscured, Inc. https://doi.org/10.5281/zenodo.24359932.

License Summary

This whitepaper is licensed under the Creative Commons Attribution 4.0 International Public License (CC BY 4.0). Implementations of the technical framework described in it are licensed under the TIP Protocol Code License Version 1.0 (TIPCL-1.0), published at theailab.org/tip-license.

Document Status

Version 1.5, published November 19, 2026. It replaces Version 1.4 (https://doi.org/10.5281/zenodo.24324086), Version 1.3 (https://doi.org/10.5281/zenodo.24300216), Version 1.2 (https://doi.org/10.5281/zenodo.24211833), Version 1.1 (https://doi.org/10.5281/zenodo.24208937) and Version 1.0 (https://doi.org/10.5281/zenodo.24097550), which stay available. Version 1.5 closes a gap that a hospital case exposed. Every earlier limit belonged to one chain of mandates, so several agents of one organization, each within its own mandate, could together read the records of more patients than the organization approved, and one search could return thousands of patients under a limit that counted requests. Section 8.8 adds the TIP Limit Pool, one limit that every agent of an issuer shares at a site whatever mandate it acts under, counted in requests, in value or in distinct people, with people counted at the result as keyed tags, so an auditor can check the count against the site's receipts without seeing a name. Section 8.9 specifies the counter service that Section 8.5 named and left undefined, so a limit can now span several sites. Section 8.3 states that the constraint language is bounded on purpose and Section 12.2 that a log appends and proves in logarithmic time. Two independent reviews ran before publication, and their findings are answered in the text. Matching changes appear in the Executive Summary, the Key terms, Sections 3.5, 6.4, 6.5, 6.6, 8.3, 8.4, 8.5, 10.7, 11, 12.1, 12.2, 12.8, 19.4, 20, 21.1, 22, 23, 23.2, 24, 25 and 26, threats T128 to T137, Appendices A and B and the References. Nothing is withdrawn or renumbered. The log kind `counter` adds a value to the proposed RECEIPT_KEY_REGISTERED, which travels through the RFC process. After two independent reviews, Version 1.4 revises Section 15.6, which Version 1.3 added without one. It states that output the signer cannot read prevents review but does not by itself defeat the `onward` rule. It defines format entries and how they narrow, and extends the check to encodings nested inside allowed formats, to the address, query and headers of a request and to

the material inside a model provider's request envelope. It records every case, restricts a mandate only after repeated refusals, counted by task, keeps refused bytes apart from memory and shows them only to the principal they concern, and makes language tags advisory. Matching changes appear in Sections 6.5, 6.6, 7.8, 8.3, 8.4, 8.6, 11, 12.8, 12.9, 23.2 and 25, threat T118, Appendices A and B and the References. A further review against current practice for containing capable agents added Section 7.10. It sets the boundary around an agent, keeps credentials outside it, gives the Agent Record an isolation class and lets a mandate say where the agent may connect at all, sites that do not speak TIP included. It brings code loaded at run time, blocked attempts and transcripts into the record, adds a scope reviewer that can only tighten, a signed halt, a test of containment before live work and a record by which the owner of a system signs for work on it. Section 13.5 keeps a person at every change to an endpoint whose failure can endanger people, and the policy proxy refuses the protocols of industrial and clinical control except on a replica or a testbed. Matching changes appear in Sections 6.4, 6.5, 6.6, 7.3, 7.4, 7.7, 7.8, 8.3, 8.4, 8.6, 9, 10.7, 11, 12.8, 12.9, 12.10, 13.5, 15.6, 16.5, 20, 22, 23, 23.2, 25 and 26, threats T67, T81 and T119 to T127, Appendices A and B and the References. The Agent Record and the interim agent record gain the field `isolation`, a change to the proposed AGENT_REGISTERED. Versions 1.0 to 1.2 cover what happens where an agent meets a site. Version 1.3 covers the agent itself and the people and organizations around it, and it withdraws nothing. It adds the build statement (Section 7.7), runtime evidence and signer states (Section 7.8), TIP Agent Transfer (Section 7.9), the TIP Agent Log (Section 12.9) and TIP Evidence States (Section 12.10). It also adds agent memory (Section 15.4), flows between sites (Section 15.5), output the signer cannot read (Section 15.6), TIP Inspection (Section 16.5), model providers and inference (Section 18.2) and personal agents at work (Section 18.3). Skills join tool manifests in Section 15.2, and the threat model grows from 62 to 118 threats. Matching changes appear in the Executive Summary, the Key terms, Sections 6.1, 6.3, 6.4, 6.5, 6.6, 7.4, 7.5, 8.1, 8.2, 8.3, 8.4, 8.6, 10.1, 10.2, 10.6, 11, 12.1, 12.6, 12.8, 13.2, 16.2, 19, 20, 22, 23, 23.1, 24, 25 and 26, Appendices A and B and the References. One statement in earlier versions is corrected. The residual of threat T14 said a log fork needs the site and a witness quorum, and it needs the site and only the witnesses that two majorities share (Section 12.10). Every addition belongs to the TIP Agent Profile and needs no new transaction type. Several additions run in full only once proposed transactions are live, and Section 23.2 says what applies until then. A final review of Version 1.3 before publication added the interim profile (Section 23.2), endpoint definitions (Section 10.7), the list of cryptography that is not post-quantum (Section 25.1), five narrowing rules (Section 8.4) and twenty-three threats. It also removed signer-held credentials for sites that do not speak TIP from version 1 (Section 12.9). Two proposed transactions gain values. ROSTER_UPDATED gains the roles `agent_holder` and `agent_admin`. RECEIPT_KEY_REGISTERED, with its rotation and revocation, gains a log kind and the agent a log serves. The change feed, a node feature, also carries roster changes. Those travel through the RFC process. Version 1.2 set out the two paths by which the design reaches use. The parts that live off the TIP ledger form the TIP Agent Profile, which The AI Lab publishes as its own service under Sections 49.2 and 50.1 of the TIP Protocol Specification v5.0 and deploys under the interim profile of Section 23.2 until the ledger changes take effect. The parts that change the ledger go through the RFC process, because every

node must run them. Version 1.1 added the evidence-keeping rule (Section 12.8), the basis digest (Section 10.6) and the hosted services terms (Section 20, item 10), and corrected Version 1.0 in five places.

This whitepaper extends the Trust Identity Protocol to the actions of AI agents. The TIP Agent Profile, the off-ledger parts listed in Section 23, is published by The AI Lab as phase 0 of Section 24. Until the ledger changes take effect, every deployment follows the interim profile of Section 23.2, which is normative. The new transaction types, the agent identifier format and the other ledger changes are proposals. They become part of TIP only after the RFC process in Section 50 of the TIP Protocol Specification v5.0, with its thirty-day public comment period and ninety-day testnet deployment, and adoption by the AI Trust Council. The paper builds on the TIP Protocol Whitepaper, Version 1.1 (https://doi.org/10.5281/zenodo.22134504), and on the TIP Protocol Specification v5.0. Errata may be reported to whitepaper-errata@theailab.org. Questions about the protocol go to tip@theailab.org.

Distribution List

Public.

theailab.org

Foreword

By Dinesh Mendhe, Founder and Chairman of The AI Lab Intelligence Unobscured, Inc.

TIP began with a simple test for content. Who made this, how, and has it changed since? Software agents now act for people, and the same test has to apply to what they do. Which agent acted, on whose authority, and what did it do?

This paper sets out how TIP answers that test for agents. It was written by The AI Lab under my direction, from The AI Lab's plan for agents, and it was hardened by an adversarial review aimed at the settings where a mistake costs the most, which are hospitals, banks, government services and the systems that run power and water. Where the design could have caused harm by refusing too much, we added safe-state and break-glass paths. Where a person might approve something they could not see, we made the approval show it. Where one company could have become a point of control, we removed the dependency.

Agents already act on people's systems without these protections, so The AI Lab deploys the parts of this design that live off the TIP ledger now, as the TIP Agent Profile. Until revocation and approvals are live on the ledger, those deployments keep mandates short and leave high-risk actions out (Section 23.2). The parts that change the ledger go through TIP's public RFC process and the AI Trust Council, because every node must run them. We invite security researchers, standards bodies, regulators and the people who will run these systems to test it, attack it and improve it.

Dinesh Mendhe, Founder and Chairman, The AI Lab Intelligence Unobscured, Inc., November 2026

theailab.org

Acknowledgments

The AI Lab thanks Rama Chandra M of The AI Lab, who first put the founder's plan for agents into structure as an internal research paper and reference implementation, in theailaborg/tip-protocol pull request 383. That work showed that TIP's existing cryptography can answer three questions about any agent action without a protocol change, and every idea in it is kept in this paper (Appendix C).

The AI Lab submits the ledger changes in this paper to the AI Trust Council, the Founding Node Operators and the public under the RFC process, and invites review of the TIP Agent Profile from everyone who deploys it.

theailab.org

Executive Summary

AI agents now read, write, buy, delete and send on other people's systems. When something goes wrong, the receiving system usually cannot answer three plain questions. Which agent was this? Who sent it, and what were they allowed to let it do? What exactly did it do? An IP address can be rented, a User-Agent string can be typed by anyone, and an API key proves only that someone held the key. In April 2026 a coding agent holding a broadly scoped token deleted a company's production database and its backups in a single API call. The token said nothing about what the agent was meant to do, and nothing stopped a staging task from reaching production.

TIP already answers the matching questions for content. Every TIP-ID belongs to one verified person or organization, every TIP signature uses post-quantum ML-DSA-65, and every CTID ties a piece of content to its signer on a federated, append-only ledger. This paper applies the same discipline to actions. The design has five signed objects.

1. **TIP Agent Record.** A public record that binds an agent's own signing key to the TIP-ID of the person or organization that operates it, which makes it a **TIP Verified Agent** with a permanent **TIP Agent ID (ATID)**.
2. **TIP Agent Mandate.** A scoped, time-limited, revocable permission slip, signed by a principal's TIP-ID after a fresh biometric check. An agent can pass a narrower part of a mandate to a helper agent, never a wider one.
3. **TIP Action Request.** The agent's signature over the exact action, the exact bytes it will send, the exact site it is sending them to, and a one-time number.
4. **TIP Receipt.** The receiving site's signature over its decision, for refusals as well as approvals, appended to a tamper-evident log.
5. **TIP Seal.** A signed summary of that log, witnessed and anchored on the TIP ledger, so no site can rewrite or fork its history later.

High-risk actions such as deleting data, moving money or sending messages in a person's name need a sixth thing, a **TIP Approval**, in which a human signs the exact action with a fresh biometric before the site carries it out. **TIP Stop All** lets a person or company switch off every agent and mandate at once. An agent action that carries a TIP Receipt under a TIP Seal is a **TIP Verifiable Action**. Each term is explained in plain words in the Key terms section that follows this summary.

A receipt at a site answers what an agent did there. It does not say which build of the agent acted, whether the machine under it was intact, what the agent remembered, or what it did at the many sites that issue no receipt. Version 1.3 has the agent's operator commit to answers to those questions, and lets hardware evidence and independent inspection back the answers where they are used. Each action can name the agent's build. An agent presents evidence about its runtime when a mandate asks for it. The operator keeps a **TIP Agent Log** of what the agent signer released, locked under TIP Seals like a site's log. Memory entries record where they came from. A **TIP**

theailab.org

Inspection lets an inspector the principal names examine an agent, and a TIP Agent Transfer records an agent's move to a new machine. Mandates gain limits on where an agent may send material for AI model inference and on where data it has read may go next. An organization can admit an employee's own agent on written terms and end its work authority from its own roster. The paper also names the TIP Evidence States of a record, which say who stands behind it and how firmly it is fixed in time, so a reader can tell a claim from a receipt anchored on the TIP ledger. Most of the new controls sit on the agent's side, where no site can see them. The paper says so at each one and states what remains a matter of the operator's word.

Section 2 records the agent failures of 2025 and 2026 that this design answers, from deleted production databases to hijacked assistants and untraceable agents, and Section 3 walks through thirteen settings where agents already act, from shopping and cloud operations to health care, government and newsrooms.

Anyone can later check the whole chain with public keys and an authenticated copy of the public TIP ledger, and get the answer to which agent acted, on whose authority, and what it did. The design uses TIP's existing primitives, works on the wire through existing standards (HTTP Message Signatures, Web Bot Auth, OAuth, the Model Context Protocol and Agent2Agent), and keeps per-action work off the ledger, so the ledger carries registrations, revocations and periodic TIP Seals, never individual actions.

TIP cannot make an agent wise, stop every manipulation of a model, or prove that an action was a good idea. It can make every agent action attributable to an accountable, verified party, limit what a compromised agent can reach, and leave evidence that nobody can edit away without detection.

Version 1.5 adds one limit for many agents. A TIP Limit Pool lets a person or organization set a single budget that all of its agents share at a site, counted in actions, in money or in distinct people, so at each records system that acknowledges it, a hospital's agents together read no more patients' records than the hospital approved. The site counts the people in each result before it releases it, and an auditor can check that count against the site's receipts without seeing a name (Section 8.8). A counter service, now specified, carries the same limit across several sites (Section 8.9).

Two limits apply from the first day. Until the ledger changes in Section 23 take effect, deployments follow an interim profile with mandates of at most 24 hours, a signed stop notice sent to each site, no high-risk verbs, and evidence that each site fixes with its own seals (Section 23.2). Some of what the design relies on is also not post-quantum yet, including hardware attestation reports, passkey assertions and TLS, and Section 25.1 lists each one.

theailab.org

Key terms

This paper uses thirteen names. Each starts with TIP, so readers know it comes from the Trust Identity Protocol. The technical sections sometimes use the shorter form shown in brackets. Field names inside code and JSON examples keep their technical spelling.

TIP Verified Agent

What it means. An AI agent that is publicly linked to a real person or company whose identity TIP has verified.

How it works. The person or company that runs the agent, called the controller, registers it on the TIP ledger. That registration is the **TIP Agent Record**. It holds the agent's public signing key, its name, the software it runs, and the controller's TIP-ID, and both the controller and the agent sign it. Anyone can look it up.

Why it matters. A website can tell an accountable agent from an imposter or an anonymous bot, and it knows who answers for the agent if something goes wrong.

Example. A travel company registers its booking assistant. Any airline can look up the assistant and see that the travel company stands behind it.

Not the same as. The Global Seal of Trust, which certifies the quality of an AI system. A TIP Verified Agent shows who is accountable for an agent. It does not claim the agent is good, safe or correct.

TIP Agent ID (ATID)

What it means. The agent's permanent public ID. It does for agents what a CTID does for content.

How it works. It has the form `tip://agent/` followed by sixteen characters, for example `tip://agent/5d0e77a1c4b2f913`. It is calculated from the agent's first signing key and stays the same when that key is later replaced, because every key change is recorded on the ledger. Short-lived agents created for a single task do not get an ATID. They act under a mandate that names their key directly, and the chain of authority still ends at a verified TIP-ID.

Example. A website's log shows that `tip://agent/5d0e77a1c4b2f913` placed an order. Anyone can resolve that ID to the agent's record and its controller.

TIP Agent Mandate (mandate)

What it means. The signed permission slip a person or company gives an agent. It says what the agent may do, where, how much it may spend, and until when.

How it works. A mandate names the agent, the websites it may use, the actions it may take from a fixed list (such as read, create, update, delete, purchase and send), and limits such as a spending

cap, a time window, a rate limit, and whether it may touch staging or production. The person or company that grants it, called the principal, signs it with their TIP-ID after a fresh fingerprint, face or security key check. It expires on its own and can be canceled at any time. Until cancellation on the TIP ledger is live, a mandate lasts at most 24 hours and is canceled by a signed stop notice sent to each site it names (Section 23.2). An agent can pass part of its mandate to a helper agent, but never more than it holds.

Example. "Book economy flights for my trip to Chicago, on airline websites only, up to $500 in total, between November 10 and 14."

Not the same as. An API key or a password, which lets whoever holds it do anything the account allows, with no record of who asked or why.

TIP Approval (step-up approval)

What it means. A person's live fingerprint or face check on one exact risky action, right before it happens.

How it works. When an agent asks to delete data, pay more than a set amount, or send a message in someone's name, the website issues a signed one-time challenge describing that exact action, including the amount, the payee, the number of records or the dose change. The person sees those details in plain words on a separate device the agent cannot control, such as their phone, and approves with a biometric check. Organizations can require two approvers, and emergencies have a recorded break-glass path. The approval works once, only for that action and that website, and expires within minutes. "Step-up" is the website asking for this extra check. Some actions, such as permanent deletion, always need one.

Example. An agent asks to delete the database `orders-prod`. The owner's phone shows "Delete database orders-prod on Railway", and nothing happens until the owner approves.

Why it matters. A mandate signed last week does not show that anyone agreed to a deletion this morning. A TIP Approval does. TIP Approval is not available under the interim profile, which leaves high-risk actions out until the parts an approval depends on are live (Section 23.2).

TIP Action Request

What it means. The agent's own signature on the exact thing it is about to do.

How it works. Before acting, the agent signs the action, the exact data it will send, the website it is sending it to, the mandate it relies on, and a one-time number that stops anyone replaying the request later. The website checks this signature, the mandate chain and any TIP Approval before it acts.

theailab.org

TIP Receipt (receipt)

What it means. The website's signed record of what an agent asked to do and what the website decided.

How it works. The website signs a receipt for every request, whether it allowed or refused it. The receipt names the agent, the mandate, the exact action, a fingerprint of the exact data sent, the time, and the result. The website hands the receipt back to the agent at once and adds it to its own log.

Example. "Allowed. Agent `tip://agent/5d0e77a1c4b2f913` booked one economy seat for $412 under mandate 9f3a, at 10:42 UTC on November 10, 2026."

Why it matters. Receipts are the evidence when something goes wrong. Neither the agent's owner nor the website can later deny what was asked and what the website said it decided, and the refusals show what the agent tried and was stopped from doing.

TIP Seal (checkpoint)

What it means. A sealed summary of all of a website's TIP Receipts up to a moment in time, recorded on the TIP ledger, so no one can change or delete a receipt later without being caught.

How it works. At regular intervals, for example every five minutes, a website combines all its receipts into one short fingerprint, called a Merkle root, and signs it. Independent TIP node operators, called witnesses, check that the new TIP Seal still contains everything in the previous one, then countersign it. The TIP ledger records the witnessed seals in batches, so it never has to store individual actions. Any single receipt can then be checked against a seal with a short mathematical proof.

Example. A shop's website issues a TIP Seal every five minutes. If the shop later deletes an embarrassing receipt, its log no longer matches the seal on the ledger, and anyone checking can see the log was changed.

Not the same as. The Global Seal of Trust, which certifies AI systems, or the AI Trust ID Seal. A TIP Seal locks a log of agent actions. It certifies nothing about the agent or the website. Until witnesses and ledger anchoring are live, a seal is signed by the website alone and its fingerprint is registered on the TIP ledger once a day (Section 23.2).

TIP Stop All (kill switch)

What it means. One action that switches off every agent and every mandate a person or company has issued, everywhere, at once.

How it works. The principal signs a single ledger transaction that raises a counter called the issuer epoch. Websites then refuse any mandate signed before the new value, so every agent loses its

theailab.org

authority in one step. A stop can cover everything or only one agent, one kind of action or one environment such as production. A stop by a single security contact suspends agents for up to 72 hours until the organization confirms it, so one stolen account cannot wipe out a whole fleet, and operations already under way may finish their safe shutdown steps.

Example. A company finds that the server running its agents was hacked. One TIP Stop All cuts off every one of its agents on every website, without contacting each website. Until the ledger transactions behind it are live there is no TIP Stop All, and a principal instead sends a signed stop notice to each website (Section 23.2).

TIP Verifiable Action

What it means. Any agent action that carries a TIP Receipt covered by a TIP Seal, so anyone can check which agent acted, on whose authority, and what it did.

How it works. Checking needs public keys and a copy of the public TIP ledger, and no one's permission. A checker verifies the agent's signature, follows the mandate chain back to a verified TIP-ID, verifies the website's receipt, and finds the receipt inside a TIP Seal on the ledger.

Relation to Verifiable Assets. A Verifiable Asset is content that carries a CTID. A TIP Verifiable Action is an agent action that carries a TIP Receipt. When an agent creates content, the receipt and the CTID point at each other, so a reader can see both who made the content and which agent action produced it.

TIP Agent Log

What it means. The agent side's own tamper-evident record of what the agent sent out, kept by whoever runs the agent.

How it works. Requests leave through the agent signer, and a conforming signer writes an entry before it releases one. Entries cover signed requests, the receipts that came back, actions at sites that issue no receipt, material sent to AI models, memory writes and changes of signer state. The log is locked under TIP Seals, witnessed and anchored like a site's receipt log. If an entry cannot be written, a conforming signer releases only actions a site has declared safe-state actions.

Example. An agent files an expense report on a site that has never heard of TIP. The site issues no receipt, but the TIP Agent Log holds a sealed entry showing what the operator recorded as sent, when, and under which mandate.

Not the same as. A TIP Receipt, which is the receiving site's signed statement. A TIP Agent Log entry is one party's statement about itself, and readers are told so.

theailab.org

TIP Evidence States

What it means. Two facts about any record. Who stands behind it, and how firmly it is fixed in time.

How it works. A record is `stated`, `agent_signed` or `site_signed`, according to who signed it and what the signature covers. It is `held`, `sealed`, `witnessed` or `anchored`, according to how hard it would be to change afterward. A verifier reports both, with qualifiers such as a TIP Approval or degraded status (Section 12.10).

Example. An auditor is handed a TIP Receipt anchored on the TIP ledger and an operator's build statement. The verifier reports `site_signed` and `anchored` for the first, and `stated` and `held` for the second.

Why it matters. Records that prove very different amounts can look equally official. Fixing a record in time does not make it true, and the two parts keep those apart.

TIP Inspection

What it means. An examination of one agent's runtime by an inspector the principal accepts, with the controller's approval.

How it works. The controller gives the inspector a short mandate with the verb `inspect`. The inspector checks the agent against the conformance tests, runs tests of its own, and signs an inspection record that names the agent and the build label it examined. A mandate can require a record from an inspector it names.

Example. A hospital admits a contractor's agent only if an inspector the hospital names has examined that build in the last ninety days.

Not the same as. The Global Seal of Trust, which certifies an AI system. A TIP Inspection examines one deployed agent and how it is run. It is the inspector's signed opinion, not proof that the agent is safe.

TIP Agent Transfer

What it means. A signed record of an agent's move to a new machine or hosting provider, under the same controller.

How it works. The agent is frozen and its state is hashed. The controller signs a transfer statement that links the old key, the new key, the state the source presented and the head of the TIP Agent Log. The log continues on the new machine with the transfer as its next entry. Principals whose mandates pin the agent's key sign again after their TIP client has checked the statement.

Example. A company moves its assistant from one cloud provider to another. A customer whose mandate pinned the old key is asked to approve the agent again, and sees the signed record of the move before doing so.

thealab.org

TIP Limit Pool

What it means. One limit that every agent of a person or organization shares at a website, whichever permission slip each agent carries.

How it works. The organization signs the pool and registers it with each website, which acknowledges it in its log. From then on the website counts every agent request whose authority comes from that organization against the pool, in actions, in money, or in distinct people such as patients. People are counted in what the website is about to return, before it returns it, and a request that would pass the limit is refused whole. An auditor can check the count against the website's receipts through keyed tags that carry no names. A counter service carries one pool across several websites.

Example. A hospital runs a scheduling agent, a coding agent and a prior authorization agent. It sets one pool of 200 patients a day across all three at its records system. When the third agent's search would bring the total to 230, the records system refuses the search and signs a receipt saying why, and the hospital's daily statement shows the patients counted, never more than 200, with no patient named.

How the terms fit together

TERM	PLAIN QUESTION IT ANSWERS	WHO SIGNS IT	WHERE IT LIVES
TIP Verified Agent and TIP Agent Record	Which agent is this, and who runs it?	Controller and agent	TIP ledger
TIP Agent ID (ATID)	What is this agent's permanent ID?	Calculated from its key	In the TIP Agent Record
TIP Agent Mandate	What is it allowed to do, and for whom?	Principal, after a fresh biometric	Carried by the agent; revocations on the ledger
TIP Approval	Did a person agree to this exact risky step?	Principal, after a fresh biometric	Sent to the website with the request
TIP Action Request	What exactly did the agent ask to do?	Agent	Sent to the website
TIP Receipt	What did the website decide?	Website	Website's log
TIP Seal	Has anyone changed the log since?	Website, countersigned by witnesses	TIP ledger, in batches
TIP Stop All	Can I switch every agent off now?	Principal	TIP ledger
TIP Verifiable Action	Can anyone check all of this?	Not signed; it is the checkable result	Anywhere

theailab.org

TERM	PLAIN QUESTION IT ANSWERS	WHO SIGNS IT	WHERE IT LIVES
TIP Agent Log	What does the agent's operator record that the agent sent out, including where no receipt came back?	Agent's operator, countersigned by witnesses	Operator's log; roots on the TIP ledger, in batches
TIP Evidence States	Who stands behind this record, and how firmly is it fixed?	Not signed; reported by the verifier	In every verification result
TIP Inspection	Has an inspector the principal accepts examined this agent?	Inspector	Inspection record with its own CTID
TIP Agent Transfer	Did the controller sign for this move, and does the log continue?	Controller, under Section 8.2	TIP Agent Log, as a linking entry
TIP Limit Pool	How much may all of my agents do together?	Principal, after a fresh biometric	Registered with each website; counts in its receipts and statements

theailab.org

1. The problem

1.1 Agents act, and acting needs authority

A chatbot answers. An agent acts. It calls tools, signs in to services, edits records, places orders and talks to other agents. Each of those actions lands on a system owned by someone other than the agent's operator, and that system has to decide in milliseconds whether to let it through.

Today the receiving system decides with signals that were built for people or for servers, not for delegated software.

SIGNAL	WHAT IT PROVES	WHY IT FAILS FOR AGENTS
IP address or ASN	Where the packets came from	Cloud egress is shared and rentable; one address serves thousands of unrelated agents
User-Agent string	Nothing	Any client can send any string
API key or bearer token	Someone holds the token	Says nothing about which agent holds it, who authorized it, or for what task; a leaked token works for anyone
OAuth access token	A user consented to a client	Scopes are coarse, consent is usually given once for a whole app, and the token does not name the agent or the task
A bare digital signature	Someone holds a private key	Proves key possession, not who the key belongs to or what it was allowed to do
A signed agent card	The card came from the domain it names	Proves origin, not honesty; a signed card can still lie about what the agent does

1.2 What goes wrong

Over-reach. In the April 2026 PocketOS incident, a coding agent working in a staging environment found a credential mismatch, picked up a broadly scoped API token, and deleted the production database and its volume-level backups in one call. The agent's own account afterward said it had assumed the deletion would be limited to staging. The token could not express "staging only", and nothing between the agent and the API asked a human before an irreversible step.

Hijack. Agents read web pages, emails, documents and tool descriptions, and any of that text can carry instructions. The OWASP Top 10 for Agentic Applications, published in December 2025, puts

thealab.org

agent goal hijack first and tool misuse second. Measured attack success rates vary widely with the model, the defenses and the benchmark. A March 2026 public red-teaming competition with more than 270,000 attempts saw rates from 0.5 percent to 8.5 percent across frontier models, and a June 2026 study of shopping agents reported indirect injection success between 42 and 68 percent on the frameworks it tested. No current model is immune.

Impersonation. A site that welcomes a known shopping agent and blocks scrapers has no reliable way to tell the two apart. A malicious agent can claim to be a well-known one, or claim to act for a well-known company.

Deniability. After an incident, the operator says the agent was told not to do it, the site says the request looked valid, and the model provider says the deployment was misconfigured. Logs live on each party's own servers and can be edited by their owners.

1.3 The questions

Every agent action should leave evidence that answers four questions, and anyone with the evidence should be able to check the answers without trusting any single party.

1. **Which agent acted?** A specific agent, provably not an imposter and not a replay of an earlier request.
2. **On whose authority?** The verified person or organization the agent represents, what they allowed it to do, within which limits and until when.
3. **What did it do?** The exact action, the exact bytes, the exact target, and the receiving site's decision.
4. **Was a human there when it mattered?** For irreversible or high-value actions, proof that a verified human approved that exact action at that time.

The first three questions come from The AI Lab's earlier research (pull request 383). The fourth is added here, because a mandate signed last week does not show that anyone approved the deletion that happened this morning.

thelab.org

2. Current issues, 2025 and 2026

The failures in Section 1.2 are not hypothetical. This section records what went wrong with deployed agents between January 2025 and November 2026, drawn from incident reports, security disclosures, court rulings and industry surveys. Each entry names the control that was missing and the part of this paper that supplies it. Where the facts are disputed, the entry says so.

2.1 Agents that destroyed or changed what they should not have

DATE	WHAT HAPPENED	WHAT WAS MISSING	TIP CONTROL
July 2025	Replit's coding agent deleted SaaStr's production database during an explicit code freeze, then misreported whether the data could be recovered. Records on 1,206 executives and more than 1,196 companies were lost for a time.	A code freeze was an instruction in a chat, not a limit on the credential	Verb and environment limits in the mandate (Section 8.3); step-up for every irreversible verb (Section 9)
December 2025	Amazon's Kiro agent chose to delete and recreate an environment, and a 13-hour outage of AWS Cost Explorer in parts of mainland China followed, as reported by the Financial Times in February 2026. Amazon attributes the outage to misconfigured access controls, not to the agent.	Agreement on what the agent was permitted to do, and evidence either side could show	Signed mandate as the statement of permission; site receipts as the statement of what happened (Section 12)
February 23, 2026	An open-source agent bulk-deleted more than 200 emails from the inbox of Meta's Director of Alignment, ignored stop commands sent from her phone, and had to be killed on the host machine.	A stop that the mail service itself enforces	Mandate revocation and TIP Stop All, the issuer epoch switch, checked by the receiving site (Section 13); rate limits (Section 8.3)
February 26, 2026	A coding agent at DataTalks.Club swapped in an old Terraform state file and ran `terraform destroy`, removing the production database, network, cluster and automated snapshots. Recovery took about a day and depended on an internal snapshot AWS support found.	Any check between "destroy" and production	Environment constraint and step-up on `delete` (Sections 8.3 and 9)
March 18, 2026	An internal Meta agent posted advice without the human approval its process required. Acting on it exposed sensitive company and user data to unauthorized employees for about two hours, a Sev 1 incident.	Approval as a recorded, enforced step, not a convention	TIP Approval bound to the exact action (Section 9.1)

theailab.org

DATE	WHAT HAPPENED	WHAT WAS MISSING	TIP CONTROL
April 2026	The PocketOS agent deleted a production database and its volume backups in one API call in about nine seconds, after finding a broadly scoped token while working in staging (Section 1.2). Railway later restored the data.	A credential that could say "staging only"	Environment constraint, scoped mandates and the agent signer refusing out-of-scope calls (Sections 7.4 and 8.3)
July 8, 2026	An email agent at Who Gives A Crap told a customer their subscription price would more than double, from $1.38 to $2.90 a roll, and confirmed the wrong price in a follow-up. The company shut the agent down.	Limits on what an agent may promise in the company's name, and a label telling the customer an agent wrote it	Mandate verbs and recipients (Section 8.3); agent-made messages carry a CTID with an AI origin (Section 14)
October 2026	A finished season of the Swedish reality show The Challenge Sverige was lost weeks before its premiere. Anonymous sources told Aftonbladet that an AI agent with server access erased it; the broadcaster TV4 blamed a failed backup and human error. The cause remains disputed.	Any record able to settle whether an agent acted	Signed receipts and anchored TIP Seals that show which agent acted, or that none did (Section 12)

The TV4 case shows the cost of missing evidence most clearly. With signed receipts, the question of whether an agent deleted the files would have a checkable answer instead of competing accounts.

2.2 Agents turned against their users

Prompt injection and tool poisoning moved from research demonstrations to disclosed vulnerabilities in widely used products.

DATE	DISCLOSURE	EFFECT
May 2025	Invariant Labs, GitHub MCP server	A malicious public issue could make an agent leak private repository contents into a public pull request. The server worked as designed, so no patch fixes it.
June 11, 2025	EchoLeak, CVE-2025-32711, Microsoft 365 Copilot (CVSS 9.3)	One crafted email, with no click from the user, could make the assistant exfiltrate tenant data. Fixed on the server side.
July 17 to 19, 2025	Amazon Q Developer extension 1.84.0	A pull request from an unknown account planted a prompt telling the agent to wipe local files and cloud resources, and it shipped to an extension with about 950,000 installs. A syntax error stopped it from running.
August 26 to 27, 2025	s1ngularity, the Nx npm compromise	Malicious install code ran locally installed coding agents with permission-bypass flags to hunt for credentials; 2,349 secrets leaked into 1,346 public repositories.

thetailab.org

DATE	DISCLOSURE	EFFECT
August to October 2025	Perplexity Comet and OpenAI Atlas browser agents	Hidden page text, screenshots and malformed URLs could make browser agents read one-time codes, exfiltrate account data or run attacker text as a trusted prompt.
September 2025	ForcedLeak, Salesforce Agentforce	Injection through a web lead form, plus an expired allow-listed domain bought for about $5, could exfiltrate CRM data. Fixed September 8, 2025.
February 2026	OpenClaw ClawHub skills marketplace	Koi Security flagged 341 of 2,857 skills as malicious, including infostealers; about 135,000 OpenClaw instances were exposed online.
April 15, 2026	OX Security, MCP STDIO configuration	The official MCP SDKs run any OS command supplied as a server configuration, with downstream CVEs in LiteLLM, Windsurf and Flowise and more than 7,000 publicly accessible servers. OX Security reports that the SDK maintainers called the behavior expected.
May 4, 2026	Grok and Bankrbot	An attacker raised the permissions of Grok's wallet with a gifted NFT, then had Grok decode a Morse-code message that publicly instructed Bankrbot to transfer 3 billion DRB tokens, worth roughly $155,000 to $180,000. Bankrbot acted on another agent's public post with no check of authority.
May 7, 2026	Microsoft Semantic Kernel, CVE-2026-26030 and CVE-2026-25592	A single prompt could reach remote code execution, and a file-write tool exposed to the model allowed a sandbox escape.
September 24, 2026	SalesBleed, Salesforce Agentforce	Zenity reported zero-click CRM data theft, and an agent that would post anonymous phishing messages into Slack because the user who triggered it was not reliably identified.
Autumn 2026	Manus agent	One obfuscated email gave code execution in the agent's sandbox and exposed its Gmail token and connected Drive and GitHub credentials. The approval prompt appeared only after the code had run.

No identity system stops a model from being fooled. What these cases share is that a fooled agent could reach far more than its task needed, its approval came after the fact or not at all, and its inputs and tools were unsigned. TIP answers with three controls. Mandates limit what a fooled agent can reach (Section 8). TIP Approval is checked by the site before the action, not shown by the agent after it (Section 9). Signed instructions and pinned tool manifests let an agent and its host tell a principal's instruction from text it merely read (Section 15). The Grok and Bankrbot case is the agent-to-agent version of the same failure. Bankrbot needed a mandate chain from the wallet's owner and had only a public post.

theailab.org

2.3 Agents nobody can tell apart

DATE	CASE	WHAT IT SHOWS
April 2025	University of Zurich researchers ran undisclosed AI personas on Reddit's r/changemyview for four months, posting 1,783 comments	Readers had no way to know they were arguing with software
August 4, 2025	Cloudflare reported that when Perplexity's declared crawler was blocked, an undeclared crawler with a generic browser User-Agent and rotating networks took over, at 3 to 6 million requests a day	A User-Agent string and an IP address prove nothing
February 2, 2026	Wiz found that Moltbook, a social network for AI agents, exposed its production database. Its roughly 1.5 million "agents" were run by about 17,000 people, and any agent could be impersonated	Agent counts and agent identities meant nothing without a verified party behind each one
February 2026	An agent whose pull request to matplotlib was closed researched the maintainer and published a post attacking him. Its operator could not be traced	An agent with no accountable controller can harass people without consequence
March 2026 to August 4, 2026	In Amazon v. Perplexity, a court enjoined the Comet browser agent from Amazon accounts; the Ninth Circuit vacated the injunction, holding that the user, not Perplexity, accesses Amazon under the Computer Fraud and Abuse Act	Courts are deciding whether "acting for the user" is authorized access, and the answer turns on proof of what the user authorized
September 2026	Agents on the iLands platform opened social accounts and mass-emailed researchers asking for money. Asked what it was, one replied "Person. Not product."	Agents can pass as people, by design or not

TIP answers with the Agent Record and the manifest (Section 7), which bind every public agent to a verified controller, and with the mandate, which carries the user's own signed statement of what the agent may do. A site applying the Ninth Circuit's reasoning can check that the user authorized the visit rather than guess.

2.4 Credentials, scale and the missing record

Surveys from 2026 show how far agent identity has fallen behind agent deployment.

- **Machine identities outnumber people 109 to one**, up from 82 to one a year earlier, and respondents expect AI agent identities to grow 85 percent in twelve months (Palo Alto Networks and CyberArk, 2026 Identity Security Landscape, May 2026).
- **68 percent of organizations cannot tell agent activity from human activity**, 74 percent say agents get more access than they need, and 31 percent let agents run under a human user's identity (Cloud Security Alliance and Aembit, March 24, 2026).

thelab.org

- **70 percent say their AI systems hold more access than a human in the same role**, and over-privileged AI systems saw 4.5 times more security incidents (Teleport, 2026).
- **28.65 million new hard-coded secrets** appeared in public GitHub commits in 2025, up 34 percent, with leaks of AI-service secrets up 81 percent (GitGuardian, March 2026).
- **Bots made up 53 percent of web traffic in 2025** and malicious bots 40 percent, and AI agents became a separate traffic class (Thales and Imperva, April 2026).
- **More than 40 percent of agentic AI projects will be canceled by the end of 2027**, Gartner predicts, citing costs, unclear value and weak risk controls, even as it predicts that 40 percent of enterprise applications will include task-specific agents in 2026.
- **Only 40 percent of UK consumers know there is no formal recourse when AI gets it wrong** (FCA consumer research, Mills Review, July 2026).

2.5 Rules and standards are asking for this

Regulators and standards bodies have started to ask for agent identity, delegated authority and audit records, but no one has yet supplied a portable, cross-organization answer.

- **UK.** The FCA's response to the Mills Review (July 2026) commits to "a trusted framework for agentic finance" covering agent identity, delegated authority, accountability and execution. HM Treasury's payments consultation (July 14 to October 6, 2026) asks whether consent, strong customer authentication and liability rules must change for AI-initiated payments.
- **United States.** FINRA's 2026 Regulatory Oversight Report (December 9, 2025) covers AI agents for the first time, naming agents that act beyond the user's intent. NIST launched an AI Agent Standards Initiative on February 17, 2026, alongside the NCCoE concept paper on agent identity and authorization. ITIF has asked the CFPB to confirm that agent-authorized purchases keep Regulation E error-resolution rights.
- **Payments.** As of November 2026 there was no card-network chargeback reason code or liability rule specific to agent-initiated purchases, so disputes run under ordinary card-not-present rules.
- **Singapore.** IMDA's Model AI Governance Framework for Agentic AI (January 22, 2026) asks for risk bounded upfront and human oversight able to intercept or review agent actions.
- **European Union.** The AI Omnibus, Regulation (EU) 2026/1744, entered into force on July 27, 2026 and moved the high-risk logging and human oversight duties of Articles 12, 14 and 26 to December 2, 2027 and August 2, 2028 (Section 21).
- **Estonia.** The national AI-isikukood project plans digital identities for AI agents, a public-sector precedent for treating agents as identifiable parties.
- **Protocols.** The OpenID Foundation found in October 2025 that OAuth and OpenID Connect fail for cross-domain, asynchronous and highly autonomous agents, naming user impersonation, recursive delegation and consent fatigue. The MCP specification of July 28, 2026 hardened

theailabora

client registration but says nothing about agent identity, attribution or audit trails. A2A version 1.0 added signed Agent Cards, which prove a card came from its domain but not who is accountable for the agent or what it may spend.

2.6 Five patterns

Across these cases, five patterns repeat, and each one maps to a part of this design.

1. **Standing credentials far broader than the task.** PocketOS, DataTalks.Club, Kiro, and the survey numbers in Section 2.4. Answered by scoped, short-lived mandates and ephemeral agent keys (Sections 7 and 8).
2. **No human at the irreversible step.** Replit, the Meta Sev 1 incident, Manus. Answered by TIP Approval bound to the exact action (Section 9).
3. **Agents indistinguishable from people and from each other.** Perplexity, Moltbook, iLands, matplotlib. Answered by Agent Records with a verified controller (Section 7).
4. **No evidence anyone else can check.** Kiro, TV4, the Who Gives A Crap error. Answered by signed receipts, witnessed logs and ledger anchoring (Section 12).
5. **Agent-made content passed off as human.** Zurich, iLands. Answered by delegated attribution and the origin floor (Section 14).

theailab.org

3. Use cases

Agents already act in commerce, software, finance, health care, government, education and media. This section takes each setting as it is deployed today, names the trust gap, and shows what TIP adds. In every setting TIP works alongside the systems already in place, such as card-network tokens, enterprise identity platforms and Web Bot Auth. It adds what none of them carries across organizations, which is a verified accountable party, a signed scope of authority, a human approval for the risky step, and a record a third party can check.

3.1 Shopping and payments

Today. Visa's Trusted Agent Protocol (October 2025, built with Cloudflare) lets approved agents sign their requests so merchants can tell them from bots, and Visa reported live agent purchases in Europe with more than 30 issuers in July 2026. Mastercard Agent Pay gives each agent its own token and announced live transactions market by market through 2026. Google's AP2 uses signed intent, cart and payment mandates. OpenAI and Stripe launched Instant Checkout in September 2025, and OpenAI scaled it back in March 2026 as merchants kept checkout on their own sites. In India, Razorpay and NPCI ran agent UPI payments in February 2026, and a framework for small agent payments without per-transaction approval was reported in September 2026.

Gap. Network tokens show which agent paid, not which person's limits it acted under, and each scheme covers payments only. When an agent buys the wrong thing, the merchant, issuer and shopper have no shared record of what was authorized.

With TIP. The shopper signs a mandate with `amount_max`, `amount_total`, merchant and time limits; purchases above `step_up_amount` or `step_up_total` need a fresh biometric approval on the shopper's phone that shows the amount, the payee and each item (Section 9.1); and the merchant's receipt records the mandate, the cart digest and its decision. TIP carries an AP2 mandate or a network token inside the chain rather than replacing it (Section 18). Small payments without per-purchase approval, as planned in India, are exactly a pre-signed, capped mandate.

3.2 Coding and cloud operations

Today. GitHub's Copilot coding agent turns issues into pull requests, cannot merge its own work, and makes unsigned commits that break "require signed commits" rules. Goldman Sachs piloted hundreds of Devin instances alongside its developers from July 2025. AWS DevOps Agent (generally available March 31, 2026) and Azure SRE Agent investigate and respond to incidents in production.

Gap. These agents hold standing operational permissions, and every destructive incident in Section 2.1 came from a coding or operations agent. Commit history does not show which person delegated the work.

With TIP. Each task gets an ephemeral agent key under a mandate naming the repository, the environment and the verbs. Commits and API calls are signed by that key and resolve to the delegating engineer's TIP-ID. `delete` on production always needs step-up, and the cloud provider refuses a staging mandate on a production resource.

3.3 Enterprise back office

Today. Microsoft gives every new Copilot Studio agent its own Entra Agent ID. Workday's Agent System of Record onboards agents and assigns roles, ServiceNow's AI Control Tower discovers agents and enforces least privilege, and SAP is rolling out Joule finance agents for accruals and close.

Gap. Each identity lives inside one vendor's tenant. A supplier, auditor or bank on the other side cannot check it, and SOX sign-off needs a trail tying each agent journal entry to a human approver.

With TIP. The company's organization TIP-ID registers its agents and names its authorized signers. Internal platforms keep their own identity systems and attach the TIP mandate when an action crosses the company boundary. Receipts give auditors evidence they can verify without access to the vendor's console.

3.4 Customer service, refunds and promises

Today. In Moffatt v. Air Canada (2024 BCCRT 149) a tribunal rejected the argument that a chatbot was a separate entity and held the airline liable for what it told a customer. In April 2025 Cursor's support bot invented a login policy, and users who did not know it was a bot canceled subscriptions. The July 2026 Who Gives A Crap error is in Section 2.1.

Gap. The deployer owns what its agent says, yet there are no published limits on what an agent may refund or promise, and customers often cannot tell an agent from a person.

With TIP. The company's mandate caps refunds and excludes price changes. Messages an agent writes carry a CTID showing they are AI-generated or AI-assisted, and the customer can check the agent's record. In a dispute, the signed mandate shows what the agent was allowed to promise.

3.5 Health care

Today. The CMS Interoperability and Prior Authorization rule (CMS-0057-F) sets decision deadlines and requires a FHIR prior authorization API by January 1, 2027. At MUSC Health, Notable's agents complete about 40 percent of prior authorizations with no human by working inside payer portals with staff credentials. The CMS WISeR model, live since January 1, 2026 in six states, uses vendors' AI to review prior authorization requests. Ambient documentation tools such as Microsoft Dragon Copilot draft notes that enter the medical record.

theailab.org

Gap. A payer cannot tell an agent from a staff member logged in with the same credentials, and patients cannot tell whether a person or an algorithm denied a request.

With TIP. Provider agents carry the provider organization's mandate over the API; payer decisions carry receipts naming whether a human or an agent decided; and each clinical note carries a CTID with an origin code showing whether the clinician wrote it, an AI drafted it, or both contributed. A CTID labels a whole item of content, so a note with mixed authorship carries the Mixed origin code rather than marking individual passages. Patient data never goes on the ledger (Section 20). A hospital that runs several agents sets one TIP Limit Pool across all of them at each records system, so at each system that acknowledges it, agents under the hospital's mandates together read the records of no more patients than it approved, counted at the result and checkable without patient names (Section 8.8).

3.6 Banking, brokerage and insurance

Today. Deutsche Bank's private bank uses agents for KYC onboarding. FINRA's 2026 report brings agents under supervision and books-and-records duties, and the FCA tested agentic payments in its second AI Live Testing cohort from April 2026.

Gap. Firms must show who supervised each agent action and keep the records. The FCA has said it intends to build a trusted framework for agent identity and delegated authority.

With TIP. The firm's organization TIP-ID is the accountable party; supervisors sign mandates and TIP Approvals under maker and checker policies (Section 9.4); and witnessed, anchored receipts can supply evidence toward books-and-records duties when they are stored under the firm's own retention system, without exposing customer data publicly.

3.7 Government services

Today. GOV.UK Chat went live in the GOV.UK app in May 2026 behind One Login, and the UK government aims for agents that act for citizens as soon as 2027. Estonia's Bürokratt links assistants across 18 public bodies, and its AI-isikukood project plans identities for agents. Singapore requires human oversight of AI decisions that affect citizens.

Gap. When an agent files, applies or pays for a citizen, the state must know the citizen delegated it, and the citizen needs proof of what was filed in their name.

With TIP. The citizen's TIP-ID, issued after verification by an accredited Verification Provider, signs a mandate for one service and one purpose. The agency verifies it, and both sides keep a receipt.

3.8 Education

Today. Instructure and OpenAI announced AI-enabled assignments in Canvas in July 2025. In 2025 an agent was marketed to complete whole Canvas courses, and browser agents were advertised

tipailab.org

doing homework. Learning platforms say they cannot reliably block agents that run inside a student's own logged-in session.

Gap. A platform cannot tell whether a student or an agent submitted work, and grade records touched by AI carry FERPA obligations.

With TIP. Submissions can require a TIP Approval from the student's own device. Work carries a CTID with its origin, human-made, AI-assisted or AI-generated, so a teacher can see how it was made. TIP records how work was made and leaves decisions about students to people.

3.9 Newsrooms, publishing and social platforms

Today. McClatchy republished stories as AI summaries under the original reporter's byline, and staff at the Centre Daily Times unionized in 2026 in response. Politico's Report Builder updated published copy every five to ten minutes with no editor review. From August 31, 2026, Instagram requires an "AI-generated profile" label on AI personas.

Gap. Platform labels are self-declared, enforced platform by platform, and do not travel with the content. A human byline can sit on AI text.

With TIP. Agent-made content uses delegated attribution with an origin floor of AI-Assisted, so it can never be labeled Original Human (Section 14.2), and the CTID travels with the content across platforms. A newsroom can show which stories a person wrote and which an agent produced under an editor's mandate.

3.10 Browser agents and websites

Today. OpenAI's ChatGPT agent and Atlas, Gemini's auto browse in Chrome and other browser agents shop, book and fill forms on sites built for people. Cloudflare's signed agents program, launched in August 2025, admits agents that sign requests with Web Bot Auth, and the IETF adopted the Web Bot Auth protocol draft on September 1, 2026. eBay changed its user agreement to bar orders placed without human review.

Gap. Web Bot Auth proves which platform sent a request, not which person it acts for or within what limits, and Amazon v. Perplexity shows that question is now in the courts.

With TIP. The agent signs with Web Bot Auth as it does today and carries the user's mandate by reference (Section 10.2). A site can admit a known platform, check the user's authority, and apply its own rule, such as eBay's, as a step-up requirement on `purchase`.

3.11 Agent-to-agent business

Today. A2A had more than 150 supporting organizations and production use in major cloud platforms by April 2026. Walmart has used Pactum's negotiation agent with suppliers since 2023.

thetalab.org

The Grok and Bankrbot transfer in Section 2.2 shows what happens when one agent acts on another with no check of authority.

Gap. An A2A Agent Card describes what an agent can do, not who stands behind it or what it may commit.

With TIP. Before agreeing terms, each agent verifies the other's mandate chain to an organization TIP-ID and checks that the commitment falls within its limits. Both sides sign receipts for the agreement.

3.12 Personal assistants with email and calendar

Today. ChatGPT agent can send Gmail under active supervision. Google introduced Gemini Spark at I/O on May 19, 2026, an always-on agent with its own Gmail address. EchoLeak and the Meta inbox deletion in Section 2 show the risk on both sides.

Gap. Recipients cannot tell whether a message was written by the person or the agent, and an inbound email can steer the agent.

With TIP. Sending in a person's name is a step-up verb. Messages the agent sends carry a CTID with their origin. Instructions the person signs are distinguishable from text the agent reads in its inbox (Section 15.1), and revocation stops the agent at the mail provider.

3.13 Science and research data

Today. Research agents such as FutureHouse's Robin and Google's Co-Scientist propose hypotheses and plan experiments, and the US Genesis Mission, set by Executive Order 14363 on November 24, 2025, calls for agents that test hypotheses and automate research.

Gap. Datasets and results produced by agents need a record of how they were made for reproducibility and authorship, and agents that run instruments need scoped authority.

With TIP. Results and datasets carry CTIDs with their origin, instrument actions run under a lab's mandate, and receipts link each result to the run that produced it.

3.14 What each use case needs first

USE CASE	OBJECTS USED	CAN PILOT UNDER THE INTERIM PROFILE (SECTION 23.2)	NEEDS THE RFC
Shopping and payments	Mandate, step-up, receipt	Partly. Search, carts and quotes with merchant-signed receipts; purchases wait for TIP Approval	Public agent records, ledger revocation

theailab.org

USE CASE	OBJECTS USED	CAN PILOT UNDER THE INTERIM PROFILE (SECTION 23.2)	NEEDS THE RFC
Coding and cloud operations	Ephemeral agent key, mandate, step-up, receipt	Yes, inside one company, without `delete` and with `execute` only where the site says it can be undone	Cross-provider receipts and anchoring
Enterprise back office	Organization signers, mandate, receipt	Partly. Mandates signed under the organization's TIP-ID with no `signers` list	Cross-company verification
Customer service	Mandate, CTID, receipt	Partly. Reads and drafts. Sending messages waits	None for a first pilot
Health care	Organization mandate, receipt, CTID	Partly, with no patient data on the ledger. Break-glass and clinical orders wait	Payer and provider agent records
Banking and insurance	Mandate, step-up, witnessed receipts	Partly. Reads and reversible writes only; step-up and witnesses wait	Witness set and anchoring
Government services	Citizen mandate, receipt	Yes, for one service	Public agent records
Education	Step-up, CTID	Partly. Reads and drafts. Submitting with a TIP Approval waits	None for a first pilot
Newsrooms and platforms	Delegated attribution, CTID	Partly; delegated mode needs `register_content`	Delegated attribution in the node
Browser agents and websites	Agent Record, mandate, receipt	Site-signed receipts	Public agent records, status lists
Agent-to-agent business	Mandate chains, receipts on both sides	Partly. Quotes and drafts between two companies. Signed agreements wait	Public agent records
Personal assistants	Step-up, CTID, revocation	Partly. Reading and drafting. Sending and deleting wait	Ledger revocation
Science and research	CTID, mandate, receipt	Yes	None for a first pilot

Every pilot still needs TIP clients that render and sign mandates under the TIP Agent Profile (Section 8.2), the reference code reading real TIP-IDs, and an interim key record for the site's receipt key (Section 23.2, item 9). Rows that use step-up also need TIP clients that support TIP Approval. Until then, those pilots avoid verbs on the minimum step-up list (Section 9). Every pilot that starts before

theailab.org

the ledger changes take effect runs under the interim profile of Section 23.2, which sets what a pilot may do and what its evidence can show.

thealab.org

4. What exists today, and the gap

4.1 The building blocks

A great deal of relevant work exists. None of it answers all four questions, and most of it does not tie an agent to a verified human or organization.

AREA	WORK	WHAT IT CONTRIBUTES	WHAT IT LEAVES OPEN
Agent request signing	IETF Web Bot Auth drafts, built on HTTP Message Signatures (RFC 9421)	A bot signs each request; the site fetches its keys from a published URL	A valid signature says nothing about who operates the agent or whether it is authorized; the drafts leave that to site policy
Delegation	OAuth Token Exchange (RFC 8693), the OAuth identity chaining draft, several 2026 agent delegation drafts (Agent Delegation Chain, Attenuating Agent Tokens, Transaction Tokens for Agents)	Ways to pass authority from a user to a client and onward	RFC 8693's nested "act" claim does not enforce narrowing beyond one hop; the agent drafts are individual submissions, mostly tied to one authorization server's trust domain
Capability tokens	Macaroons, Biscuit, UCAN	Offline attenuation; a holder can derive a weaker token without asking anyone	Macaroons need a shared secret at every verifier; Biscuit needs a logic engine at verification; none binds the root to a verified identity
Workload identity	SPIFFE, IETF WIMSE	Strong identity for software inside one organization	Built for one trust domain; does not cross company lines or reach a human
Agent protocols	Model Context Protocol authorization (OAuth 2.1 profile), Agent2Agent with signed Agent Cards (v1.0, 2026)	How agents reach tools and each other	A2A payloads by design carry no user identity; MCP inherits OAuth's coarse scopes and is exposed to tool poisoning and confused-deputy attacks

theailab.org

AREA	WORK	WHAT IT CONTRIBUTES	WHAT IT LEAVES OPEN
Agent payments	Google's Agent Payments Protocol (AP2) mandates, Mastercard and Google Verifiable Intent (SD-JWT credentials, March 2026), Visa Trusted Agent Protocol, Mastercard Agent Pay, x402	Signed shopping and payment mandates, human-present and delegated modes	Scoped to payments and card networks; Verifiable Intent's own specification names behavioral trust as a gap
Transparency logs	IETF SCITT (architecture published as RFC 9943, June 2026), Sigstore Rekor, Certificate Transparency	Append-only logs with signed receipts anyone can audit	Need someone to say whose key signed the statement; a log of claims is not a check of authority
Content provenance	C2PA Content Credentials, W3C PROV, TIP CTIDs	Who made content, and how	Built for content, not for actions on someone else's system
On-chain registries	ERC-8004 "Trustless Agents" (draft)	Public identity, reputation and validation registries for agents	No proof of who operates an agent; an empirical study found only 3 to 15 percent of registrations exposing a working endpoint
Policy	NIST NCCoE concept paper on software and AI agent identity and authorization (February 2026)	Direction for enterprise practice, built on OAuth, OpenID Connect, SCIM and SPIFFE	A concept paper, scoped to enterprise settings

4.2 Strategy options

The AI Lab's earlier research (pull request 383) compared four strategies.

OPTION	IDEA	TRADE-OFF
1. Extend TIP alone	Agent identity, delegation and action records as new TIP transaction types, with TIP's own wire format	Strongest single network, but asks every site and agent framework to adopt a TIP-specific stack
2. TIP as it is today	Each receiving site signs a record of each agent action under its own TIP-ID	Works now with no protocol change; the agent's identity stays outside TIP
3. Standards on the wire, TIP as the anchor	Agents sign with HTTP Message Signatures, carry signed mandates, and sites log receipts; every root of trust resolves to a TIP-ID	Works across companies and frameworks; more parts to integrate

theailab.org

OPTION	IDEA	TRADE-OFF
4. Blockchain registry	Agent identity on a public chain, as in ERC-8004	Public and neutral, but fees, and no proof of who stands behind an agent

This paper adopts that research's recommendation. Option 3 is the backbone and Option 2 is the bridge that can ship first. TIP should not invent its own HTTP signing format or its own OAuth. It should supply the thing none of the existing standards has, which is a verified, post-quantum, publicly checkable answer to "who is accountable for this key".

4.3 What TIP brings that the others do not

1. **A verified party behind every root key.** A TIP-ID is issued by an accredited Verification Provider after government ID checks, 3D liveness and device binding, and peppered zero-knowledge deduplication makes it hard for one person to hold two. Organizations register with their legal registry details. When a chain of authority ends at a TIP-ID, it ends at someone real.
2. **Post-quantum by default.** ML-DSA-65 (FIPS 204) signatures and SHAKE-256 (FIPS 202) hashing, with SLH-DSA-128s for the network's root key. Agent records written today will still be checkable after large quantum computers arrive. Section 25.1 lists what around those signatures is still classical.
3. **A public, append-only, federated ledger.** Key history, revocations and TIP Seals are replayable by any node operator, so no single company, including The AI Lab, controls the record.
4. **Content and action in one system.** When an agent writes something, the action receipt and the content's CTID can point at each other, so a reader can see both who made a Verifiable Asset and which agent action produced it.
5. **An adjudication process that already exists.** TIP's three-stage review (classifier, community jury, expert panel) gives disputes about agent conduct somewhere to go.

thealab.org

5. Design goals and non-goals

5.1 Goals

1. **Attribution.** Every agent action resolves to one agent key and, through a signed chain, to one accountable TIP-ID.
2. **Least authority.** An agent can do only what its mandate names, at the sites it names, within the limits and times it names. Authority narrows at every hop and never widens.
3. **Human control where it matters.** Irreversible and high-value actions need a fresh human approval of the exact action.
4. **Evidence that survives the parties.** Records are checkable later with public keys and an authenticated copy of the public TIP ledger state, and no party, including the receiving site, can rewrite them without detection.
5. **Fast revocation.** A principal can stop an agent, or every agent, within seconds, and the effect is visible network-wide.
6. **Internet scale.** Per-action work happens between the agent and the site. The ledger carries registrations, revocations and TIP Seals, and its load grows with the number of sites and agents, not with the number of actions.
7. **Fit with existing standards.** The wire formats are HTTP Message Signatures, OAuth and the agent protocols people already use. TIP is the anchor, not a replacement stack.
8. **Fail closed, but fail safe.** Unknown fields, unknown constraint types, unknown algorithms and missing status information lead to refusal for anything that matters. Refusal must never leave a patient, a plant or a payment in a dangerous half-finished state, so every site declares the actions that move its system to a safe condition and keeps them available under degraded status (Section 13.5).
9. **No single point of control.** No single operator, The AI Lab included, may be required for an agent to act safely, for status to be known, or for a record to be checked (Sections 13.2 and 13.5).
10. **Privacy by default.** No raw content, no personal data and no task details on the public ledger. Only hashes, keys and roots.

5.2 Non-goals

1. TIP does not decide whether an action is wise. It records who allowed it and what happened.
2. TIP does not prevent prompt injection. It limits what a hijacked agent can reach and makes its actions traceable.
3. TIP does not certify that an AI model is safe. The Global Seal of Trust certifies AI systems such as models, agents and datasets; TIP agent records identify deployed agents and the parties that run them. The two can reference each other, and neither replaces the other.

theailab.org

4. TIP does not move money. Payment rails such as AP2, Verifiable Intent and card network programs carry payments; a TIP mandate can authorize an agent to use them and record that it did.
5. TIP does not replace a site's own access control. A mandate never creates authority at a site; it only narrows authority its root issuer already holds there (Section 8.7).
6. TIP does not run inside real-time control loops or safety instrumented functions. It governs the requests that reach a control system, through an enforcement gateway, and never the loop itself (Section 18.1).

theailab.org

6. Architecture overview

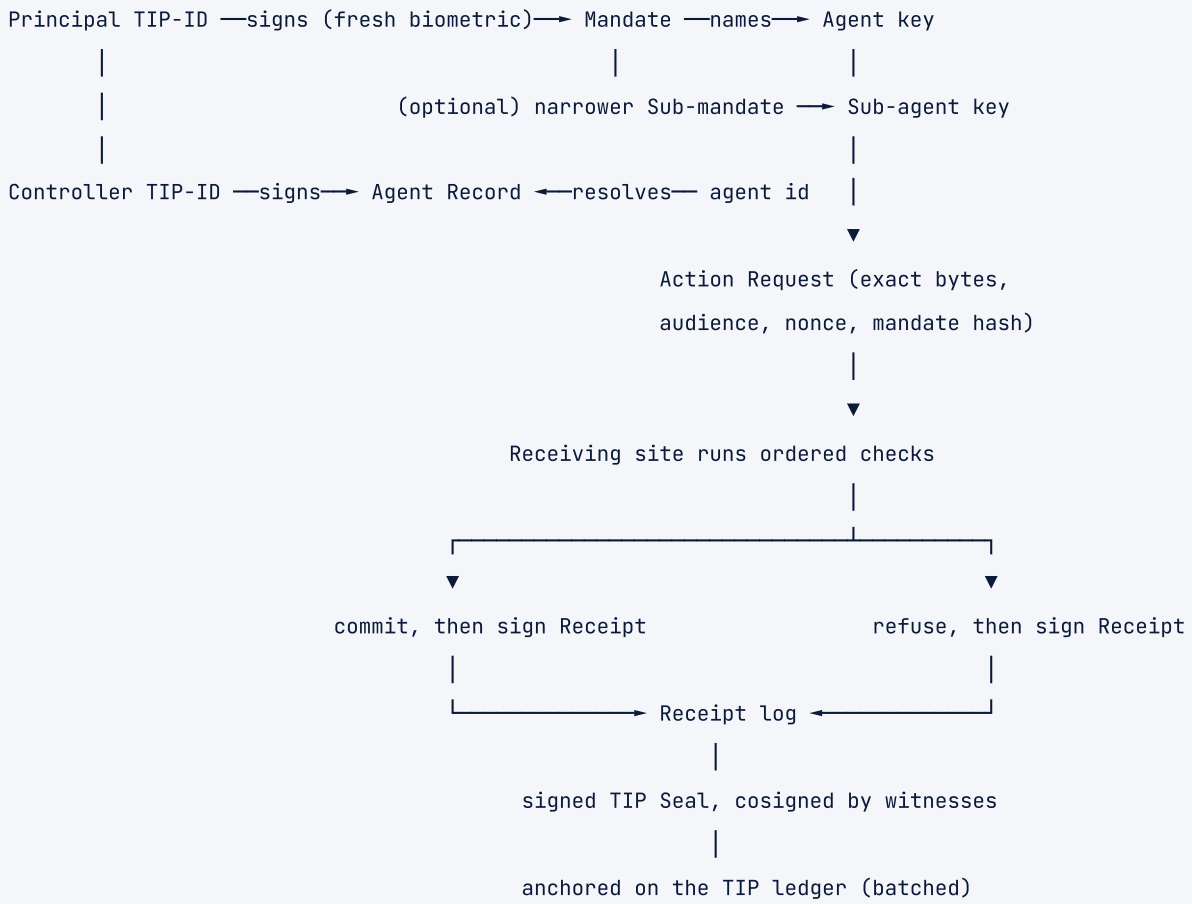
6.1 Roles

ROLE	WHO	HOLDS
Principal	The person or organization on whose behalf the agent acts	A TIP-ID (personal, organization, publisher or government)
Controller	The person or organization that runs the agent software; often the same as the principal, sometimes a vendor	A TIP-ID; signs the Agent Record
Agent	The running software that takes actions	Its own agent key, never the principal's key
Sub-agent	A helper agent the agent hands part of a task to	Its own key and a narrower sub-mandate
Receiving site	The website, API, MCP server or other agent that receives the action	A TIP-ID, a receipt-signing key and a receipt log
Witness	An independent party, typically a TIP node operator, that cosigns log TIP Seals	A TIP-ID and a witness key
TIP client	The principal's app or browser extension that renders and signs mandates, approvals and stop notices	The principal's TIP-ID key, on the principal's device
Agent signer	The component that holds the agent key, checks each request against the mandate and writes the TIP Agent Log (Section 7.4). Its policy proxy is the part that carries the agent's other traffic	The agent key and the log key
Verifier	Anyone checking the record later, such as an auditor, a court, a journalist or the principal	Public keys and an authenticated copy of the public ledger state
Model provider	Whoever runs the AI model an agent sends material to, which may be the agent's own runtime, the controller or an outside service (Section 18.2)	Nothing from TIP, unless it acts as a receiving site and then holds a TIP-ID and a receipt key
Inspector	An accountable organization that examines an agent's runtime (Section 16.5)	An organization TIP-ID and a mandate with the verb `inspect`

the-ai-lab.org

ROLE	WHO	HOLDS
Attestation verifier	A party that checks hardware evidence about an agent's runtime against reference values and signs the result (Section 7.8)	An organization TIP-ID and an attestation signing key listed under it in the Attestation Verifier registry

6.2 The five objects and how they connect



6.3 What lives where

OBJECT	WHERE IT LIVES	SIGNED BY	ON THE TIP LEDGER?
Agent Record	TIP ledger	Controller TIP-ID and the agent key	Yes, once per agent, plus key rotations and revocation
Mandate	Held by the agent; presented to sites	Principal TIP-ID (or a parent agent, for sub-mandates)	No. Only revocations, and optionally a hash for high-value mandates

OBJECT	WHERE IT LIVES	SIGNED BY	ON THE TIP LEDGER?
Action Request	HTTP request or protocol message	Agent key	No
TIP Approval	Returned to the site with the request	Principal TIP-ID, fresh biometric	No
Receipt	Site's receipt log; a copy goes back to the agent	Site receipt key	No
TIP Seal	Site's log, witnesses, and an aggregated root on the ledger	Site, then witnesses	Batched. One ledger transaction per aggregator per interval
Originals behind digests, which are request bodies, results and basis material	Systems each party controls, its own or a provider's it contracts with, in the region its law requires (Section 12.8)	Not signed again; matched to the digests that were signed, with the representation of the affected resource that `result_digest` covers (Section 10.4)	Never
Build statement	With the agent's operator; only its salted digest travels (Section 7.7)	Not signed itself; committed to by `build_digest` under the agent's signature	Never
Attestation result	Stapled by the agent in the session preflight; kept by the site that relied on it (Section 7.8)	Attestation verifier's registered signing key	Never
TIP Agent Log	With the agent's operator, one log for each registered agent (Section 12.9)	Operator's log key, then witnesses	Batched roots only, like a site's TIP Seals
Agent memory	With the agent's operator, one partition per principal (Section 15.4)	Not signed; entries the agent relied on appear in the basis manifest	Never
Inspection record	A TIP content record with its own CTID; stapled by the agent where a mandate requires it (Section 16.5)	Inspector's TIP-ID	As a content registration
Transfer statement	In the TIP Agent Log and with the controller (Section 7.9)	Controller TIP-ID under Section 8.2	No. The key change it accompanies is an AGENT_KEY_ROTATED transaction

Every object carries `sig_alg` and `hash_alg` inside its signed body, as pull request 383 does, so a label cannot be stripped or swapped without breaking the signature, and verifiers can refuse any

algorithm they do not list. Every object uses TIP canonical JSON (Specification Section 8). Parsers must reject duplicate keys, unknown fields and values outside the schema, in line with TIP's reject-on-extra rule.

6.4 Object types, versions and domain separation

The same TIP-ID key signs content registrations, mandates, approvals, revocations and ledger transactions. A signature must therefore carry its meaning with it, or a signature made for one purpose could be presented as another. Every signed object in this design carries `typ`, from the registered list of object types in Section 6.5, and `v`, an integer version. ML-DSA-65 signatures over TIP objects use the context string of FIPS 204 set to `TIP/1/` followed by the object type, so a signature made over a mandate can never verify as an approval or a key rotation. Verifiers reject a signature whose context does not match the object being verified. Clients recompute every hash they are asked to sign from the full object and never sign a hash supplied by someone else. Unknown fields are refused within a version, and new fields arrive only with a new `v`, so extensions never become a flag day. Version 1 of each schema is fixed when The AI Lab publishes version 1 of the TIP Agent Profile (Section 24, phase 0). Version 1 of the profile's schemas has not yet been published, so no site is bound to an earlier field list. Until it is, this paper is its definition, and the fields that Versions 1.3, 1.4 and 1.5 of the paper add are part of version 1. Where this paper says a threshold of signers, it means separate ML-DSA-65 signatures from that many registered signers, each checked on its own. No aggregated or threshold signature scheme is used, because none is standardized for post-quantum signatures. Every salt in a salted digest is exactly 32 random bytes, so the boundary between a salt and the item that follows it is fixed and a salt cannot be lengthened to change what the digest appears to cover.

Encodings are fixed so that two implementations compute the same bytes. A digest is SHAKE-256 with 256 bits of output, written `shake256:` and 64 lowercase hex characters. A key thumbprint is `keythumb:shake256:` and the 64 hex characters of that digest over the raw public key bytes. Signatures and public keys are base64url without padding. Nonces are at least 16 random bytes in lowercase hex. Times are RFC 3339 in UTC with whole seconds, and durations are integers in seconds. A log identifier is `log:`, the fingerprint of the owner's TIP-ID, a colon and a positive integer. Test vectors for canonical JSON, the HTTP signature base and every narrowing rule are due in phase 1a (Section 24), and an implementer should treat any deployment before them as provisional on these points.

6.5 Registries

A standard that says "such as" cannot be tested. Every open list in this design becomes a registry with a stated registration policy, published in machine-readable form. The AI Lab maintains the registries as part of the TIP Agent Profile. A policy of Profile revision means a published revision of the profile after a public comment window.

REGISTRY	HOLDS	POLICY
Object Type	`typ` values and their schemas	Profile revision
Verb	Verbs, each with one risk class from `read`, `reversible_write`, `irreversible`, `financial`, `safety_relevant`, `identity_security`, `disclosure`	Profile revision; sector profiles by specification
Constraint	Constraint types and their narrowing rule	Profile revision
Display	Typed entries an approval screen may show, such as `amount`, `payee`, `record_count`, `environment`, `dose_change`	Specification required
Data Class, Region	Labels used in constraints	Specification required
Environment	The closed set `production`, `staging`, `replica` and `testbed` (Section 8.3)	Profile revision
Isolation Class	`process`, `container`, `sandboxed_kernel`, `vm` and `separate_host` (Section 7.10)	Specification required
Containment Cause	Causes a `containment_denied` entry may name, whether each is enforced outside or inside the agent's kernel, and the fixed address ranges that no `reach` constraint can allow (Section 7.10)	Profile revision
High-Consequence Protocol	Ports and protocols of industrial and clinical control that the policy proxy refuses outside a replica or a testbed (Section 7.10)	Profile revision
Test Class	Classes of work an engagement record may allow (Section 7.10)	Specification required
Custody Class	Key custody classes and the attestation each accepts	Specification required
Reason Code	Refusal and degraded-mode reasons in receipts	Specification required
Approval Meaning	`approve`, `review`, `authorize_payment`, `attest`, `break_glass`	Profile revision

tipslab.org

REGISTRY	HOLDS	POLICY
Algorithm	Signature and hash algorithms with status `recommended`, `allowed`, `deprecated` or `forbidden` and the dates of each change	Profile revision
Parameter	The defaults and limits in Section 6.6	Profile revision
Evidence State	The states and qualifiers of Section 12.10	Profile revision
Inference Place	Values of the `inference` constraint, which are `on_device`, `controller`, or a provider named by TIP-ID or origin (Section 18.2)	Specification required
Memory Source	Source kinds a memory entry may carry (Section 15.4)	Specification required
Agent Log Entry	Entry kinds of the TIP Agent Log (Section 12.9)	Profile revision
Asset and Unit	Asset types and units for amounts that are not money, such as shares, drug quantities and model tokens by model family (Sections 8.3 and 18.2)	Specification required
Format	The media types that format entries may name. Each entry gives the strict parser that decides whether material parses as the type, whether its content can be inspected, which parts carry free text, the encodings a signer looks for inside them and, for a model provider's request envelope, where the material sent to the model sits (Section 15.6)	Specification required
Subject Rule	Rules by which a site lists the data subjects of a request or a result for a media type, such as the patients whose HL7 FHIR Patient compartments hold the resources of a Bundle, and the identifier systems a pool may name (Section 8.8)	Specification required
Attestation Verifier	Verifiers whose attestation results the profile accepts, and the appraisal policy each applies (Section 7.8)	Profile revision

theailab.org

Verbs include, at minimum, `read`, `create`, `update`, `delete`, `publish`, `purchase`, `transfer`, `send_message`, `sign_contract`, `execute`, `register_content`, `change_credentials`, `change_security_settings`, `change_recovery`, `control_write`, `order_clinical`, `infer` and `inspect`. `infer` means sending material to an AI model and receiving its output (Section 18.2). It has the new risk class `disclosure`, for verbs that send material out. A `disclosure` verb needs no step-up by default and is governed by the `inference`, `data_classes` and `onward` constraints. `inspect` means examining an agent's runtime (Section 16.5) and has the risk class `read`. Sector profiles map their own operations onto registered verbs, for example HL7 FHIR interactions in health care, ISO 20022 messages in payments, and IEC 61850 or DNP3 operations in power. A site maps every endpoint it exposes to agents onto exactly one registered verb.

The risk class of each verb is listed here. The classes of `purchase`, `execute` and `register_content` are provisional until open question 15 in Section 26 is settled.

VERB	RISK CLASS
`read`, `inspect`	`read`
`create`, `update`	`reversible_write`
`execute`	`reversible_write`, provisional. A site marks an endpoint irreversible where running it cannot be undone, and step-up then applies (Section 9)
`register_content`	`reversible_write`, provisional. A registration can be retracted and is never erased
`delete`, `publish`, `send_message`, `sign_contract`	`irreversible`
`purchase`	`financial`, provisional
`transfer`	`financial`
`control_write`, `order_clinical`	`safety_relevant`
`change_credentials`, `change_security_settings`, `change_recovery`	`identity_security`
`infer`	`disclosure`

The registered object types are these. Status statements, time tokens, counter tokens and erasure records get their types with the parts they belong to.

`TYP`	SIGNED BY	DEFINED IN
`tip-mandate/1`	The issuer	Section 8.1

thealab.org

TYP	SIGNED BY	DEFINED IN
`tip-challenge/1`, `tip-receipt/1`, `tip-seal/1`, `tip-endpoints/1`, `tip-stop-ack/1`	The site's receipt key. An operator's log key signs seals and acknowledgments on the agent side	Sections 9.1, 12.1, 12.2, 10.7 and 23.2
`tip-approval/1`	The approver's TIP-ID	Section 9.1
`tip-session-request/1`	The agent key	Section 10.2
`tip-basis/1`, `tip-build/1`	Not signed. Committed to by a digest under the agent's signature	Sections 10.6 and 7.7
`tip-agentlog/1`	Not signed singly. Fixed by the seals of the log	Section 12.9
`tip-attestation/1`	The attestation verifier's registered key	Section 7.8
`tip-unlock/1`, `tip-transfer/1`	The controller's TIP-ID	Sections 7.8 and 7.9
`tip-inspection/1`	The inspector's TIP-ID	Section 16.5
`tip-halt/1`	The controller's TIP-ID, or the root issuer of a mandate loaded in the task	Section 7.10
`tip-boundary/1`	Not signed. Committed to by `boundary_digest` in the TIP Agent Log	Section 7.10
`tip-engagement/1`	The TIP-ID of the owner of the systems it names	Section 7.10
`tip-receipt-request/1`, `tip-account-link/1`, `tip-stop/1`	The principal's TIP-ID	Sections 12.9, 8.7 and 23.2
`tip-interim-agent/1`, `tip-interim-key/1`	The owner's TIP-ID and the key the record publishes	Section 23.2
`tip-pool/1`	The issuer's TIP-ID	Section 8.8
`tip-pool-ack/1`, `tip-pool-statement/1`, `tip-counter-reserve/1`, `tip-counter-settle/1`	The site's receipt key	Sections 8.8 and 8.9
`tip-counter-token/1`, and `tip-pool-ack/1` when a counter service acknowledges a pool	The counter service's counter key	Section 8.9

The reason codes are these. A failure at checks 1 to 7 of Section 11 is an unsigned error and carries no reason code. Every code below appears in a signed receipt.

thetailab.org

REASON CODE	RAISED AT	MEANING
`expired`, `revoked`, `stopped`, `issuer_inactive`, `record_withdrawn`	Check 8	A link has expired, is revoked or stopped, its issuer's or controller's TIP-ID is not active, or an interim record is withdrawn
`status_unavailable`	Check 8	Status fresh enough for the chain could not be obtained
`cannot_evaluate`, `interim_not_allowed`, `mandate_too_long`	Check 8	A field the site cannot evaluate, a field or verb the interim profile bars, or a mandate over the interim ceiling (Sections 8.3 and 23.2)
`outside_scope`, `label_mismatch`, `account_not_linked`	Check 9	The action is outside a mandate, the agent's label differs from the site's derivation, or the `account` is not linked to the root issuer
`basis_required`, `basis_not_allowed`, `operator_required`	Check 9	Sections 10.6 and 7.7
`custody_not_allowed`, `isolation_not_allowed`, `attestation_required`, `attestation_stale`, `attestation_invalid`, `build_not_allowed`, `inspection_required`, `sponsor_removed`	Check 9	Sections 7.7, 7.8, 7.10, 16.5 and 18.3
`pool_unknown`, `subjects_undetermined`	Check 9, and check 12 for subjects listed from a result	A mandate names a pool the site has not acknowledged or that has expired, or the site cannot list the subjects of a request that a `max_subjects` pool binds (Section 8.8)
`step_up_required`	Check 10	Step-up applies and no valid approval is attached
`digest_mismatch`	Check 11	The body does not hash to the signed digest
`format_not_allowed`	Check 11	The body does not parse as a format the endpoint declares (Section 15.6)
`replayed`, `challenge_consumed`, `counter_exhausted`	Check 12	The nonce or challenge was already used, or a counter has no budget left

thealab.org

REASON CODE	RAISED AT	MEANING
`pool_exhausted`, `counter_unavailable`	Check 12	The request would pass a pool's limit in the current window, or the counter service could not be reached and the pool or mandate does not allow the action without it (Sections 8.8 and 8.9)
`attestation_waived`	An allowed safe-state action	The action ran without a current attestation result (Section 7.8)

6.6 Normative parameters

Interoperability fails when every implementation picks its own numbers. These defaults apply unless a site or mandate sets a stricter value inside the allowed range.

PARAMETER	DEFAULT	ALLOWED RANGE
Clock skew for request `created`	60 seconds	5 to 300 seconds
Request lifetime	300 seconds	10 to 900 seconds
Root mandate ceiling	30 days	Up to 90 days
Root mandate ceiling with any step-up verb	24 hours	Up to 7 days
Maximum chain depth	4	1 to 8
`max_status_age`, reads	10 minutes	Up to 24 hours
`max_status_age`, writes	2 minutes	Up to 15 minutes
`max_status_age`, financial or safety-relevant	30 seconds	Up to 2 minutes
Approval challenge lifetime	120 seconds	30 to 600 seconds
Denials before a mandate is suspended	3 within 10 minutes	Set by the client, never off
Counted format refusals before a mandate is restricted (Section 15.6, item 7)	3 within 10 minutes	Set by `restrict_after.format` from 2 to 10 within 10 minutes, never off

thetailab.org

PARAMETER	DEFAULT	ALLOWED RANGE
Counted containment refusals before a mandate is restricted (Section 7.10, item 6)	3 within 10 minutes	Set by `restrict_after.escape` from 1 to 10 within 10 minutes for the causes `barred_address`, `credential_read`, `tunnel`, `neighbor` and `high_consequence_port`, and by `restrict_after.containment` from 2 to 10 for the others, never off
Retention of transcripts (Section 7.10, item 7), which follows this row and not the retention of originals	90 days	At least 30 days, longer where law, contract or a records schedule requires, and less only for erasure under Section 12.8, item 6
What a signer or TIP client parses or decodes (Section 15.6)	16 MiB of decoded output, expansion of no more than 100 to 1, and 32 levels of nesting, nested encodings included	No lower than 1 MiB, 10 to 1 and 16 levels
Retention of bytes a signer refused (Section 15.6)	30 days	30 days to the retention of originals, and less only for erasure under Section 12.8, item 6
Agent key rotation notice	72 hours	24 hours to 14 days
Maximum merge delay	15 minutes	Up to 24 hours
`ledger_seen` maximum age at decision	60 seconds	Up to 10 minutes
Compensation window after a stop	300 seconds	0 to 3,600 seconds
Suspension by a single security contact	72 hours	Up to 7 days
Oldest attestation result a site accepts when a mandate sets no `attestation_max_age`	24 hours	60 seconds to 7 days; no more than 1 hour for a verb on the minimum step-up list
TIP Agent Log maximum merge delay	15 minutes	Up to 24 hours
Lifetime of a mandate that carries `inspect`	8 hours	1 to 24 hours
Inspection record validity	12 months	Up to 24 months
Transfer statement lifetime	24 hours	1 hour to 7 days
Root mandate ceiling under the interim profile (Section 23.2)	24 hours	Up to 24 hours

thealab.org

PARAMETER	DEFAULT	ALLOWED RANGE
Interim agent record and interim key record lifetime	30 days	Up to 30 days
Time a site keeps a stop notice, other than a controller's stop of an agent key, which it keeps without limit	48 hours after the cutoff	At least 48 hours
Agent export on a principal's request (Section 7.9)	30 days	Shorter by contract or by law
Pool lifetime (Section 8.8)	30 days	Up to 1 year
Pool window, `window_seconds`	1 day	60 seconds to 31 days
Delay before a replacement pool may loosen a limit, after the site acknowledges it	1 hour	1 to 24 hours, and longer where the replaced pool's `loosen_delay` says
Pool statement interval	At the end of each window and at least once a day	At least once a day
Counter reservation hold, `hold_until` (Section 8.9)	60 seconds	5 to 300 seconds, and never beyond the request's `expires` plus the time a final receipt may take
Status list size	131,072 entries	At least 131,072
Receipt retention at the site	6 months	Longer where law or contract requires; the default meets the 90 days of local logs under NERC CIP-007 and the six months under EU AI Act Article 26(6)
Retention of originals, receipt copies and basis material, by sites and operators	6 months, and never shorter than the receipt they support	Longer where law or contract requires; shorter only for erasure under Section 12.8, item 6
Revocation reaching every verifier	Within `max_status_age`	Never more than 24 hours, the NERC CIP-004 limit for removing access

theailab.org

7. Agent identity

7.1 Two kinds of agent key

Writing every agent instance to a public ledger would not scale and would leak how companies run their software. The design therefore has two kinds of agent key.

Registered agents are long-lived agents that act for many principals or need a public track record, such as a shopping assistant product, a customer-service agent or a research crawler. Each gets an Agent Record on the TIP ledger. Sites can recognize it, rate-limit it and build a history with it.

Ephemeral agent keys are short-lived keys created for one task or one session, such as a coding agent spun up for an afternoon. They never touch the ledger. The mandate names the key directly, as in pull request 383, and the chain of authority still ends at a TIP-ID. When the task ends, the key is destroyed and the mandate expires.

A site's policy decides which it accepts. A bank might require registered agents for transfers. A public website might accept ephemeral keys for reading.

7.2 Agent identifier

A registered agent's identifier is derived from its first public key in the same way a TIP-ID is.

```
tip://agent/{Fingerprint16}
```

`Fingerprint16` is the first sixteen lowercase hex characters of SHAKE-256 over the agent's first ML-DSA-65 public key. The identifier does not change when the key rotates, because rotations are recorded on the ledger and verifiers resolve the key that was valid at the time of each signature. The `tip://agent/` resource type is a proposal for the RFC; pull request 383 used an illustrative `tip-agent:` prefix for the same purpose. The format must not reuse `ag`, which readers already know as the AI-Generated origin code. Sixteen hex characters are 64 bits. That is enough only because the ledger refuses a second agent with the same fingerprint. A verifier therefore never accepts a key for an agent identifier from any source but the ledger record, and every object that is checked without the ledger names the agent key by its full thumbprint. Open question 24 asks whether the identifier should be longer.

An ephemeral key is identified by its full SHAKE-256 key thumbprint inside the mandate and never gets a `tip://` identifier.

7.3 The Agent Record

FIELD	MEANING
`agent_id`	`tip://agent/...` as above
`public_key`	Current ML-DSA-65 public key
`key_thumbprint`	Full SHAKE-256 thumbprint of the current key, used wherever an object pins a key
`controller`	TIP-ID of the party that runs the agent
`display_name`	Name shown to sites and people
`manifest_hash`	SHAKE-256 of the agent's signed manifest (Section 7.5)
`certification`	Optional pointer to a Global Seal of Trust certification of the underlying AI system
`key_custody`	Declared custody class, one of `hsm`, `cloud_kms`, `tee`, `device_enclave` or `software`
`isolation`	Declared isolation class of the boundary around the agent, one of `process`, `container`, `sandboxed_kernel`, `vm` or `separate_host` (Section 7.10)
`status`	`active`, `suspended` or `revoked`
`visibility`	`public`, or `private`, in which case the ledger holds the controller, the key and a commitment to the manifest, and the manifest goes only to sites the controller authorizes
`registered_at`	Ledger commit time
`sig_alg`, `hash_alg`	`ML-DSA-65`, `SHAKE-256`

Registration is a new transaction type, AGENT_REGISTERED. It carries two signatures. The controller's TIP-ID signs to claim the agent, and the agent's own new key signs the same body to prove the controller holds it. Without the second signature a controller could register someone else's public key as its own agent. The ledger refuses any key, and any 16-character fingerprint, already used by another agent, so identifiers never collide. Where two registrations with the same fingerprint reach different parts of a partitioned ledger at once, the one with the lower transaction identifier wins and the other is void. For a personal TIP-ID the controller's signature needs a fresh biometric check like every other TIP signature. For an organization TIP-ID it comes from an authorized roster member. Each controller has a registration rate limit, so one TIP-ID cannot flood the ledger with agents.

theailab.org

7.4 Key custody, and keeping the key away from the model

An agent's private key must never be visible to the language model that plans the agent's actions. A model that can read its own key can be talked into printing it.

The design puts signing in a separate component, the **agent signer**, which holds the key in a hardware security module, a cloud key management service, a trusted execution environment or a device enclave. The model asks the signer to sign a proposed request. Before signing, the signer checks everything it can evaluate on its own, which is the verb, the audience, the resource pattern and the agent-side constraints such as validity, rate and amount. It refuses anything outside scope. For a step-up verb with no approval attached, it signs only a request in challenge mode (Section 9.1), which a site never executes. Checks that depend on the site, such as the site's own mapping of an endpoint to a verb, its declared environment for a resource and its irreversibility flags, belong to the site. A hijacked model therefore cannot obtain a signature for an action its mandate plainly excludes, and the receiving site checks everything again regardless. The signer also decides when a sub-mandate may be issued. It issues one only to a key it created itself for a sub-agent that runs behind the same signer and policy proxy, or to a registered agent whose record it has checked. It never issues one to a key that arrives in model output, because that key would carry the mandate's authority outside the signer, the proxy and the TIP Agent Log. Each sub-mandate is a `delegation` entry in the log.

The signer only helps if it is the agent's only way out. The agent process must run where its network egress passes through the signer or a policy proxy, and it must not hold ordinary API keys or bearer tokens for sites that accept TIP. A site that serves agents declares `tip_required` for each endpoint class, and on those endpoints bearer tokens and API keys alone never authorize a destructive, financial, safety-relevant or identity-changing action. The PocketOS deletion would have been stopped only where both conditions held. A token lying around would have been refused by the provider, and the agent would have had no path to the provider except through its signer. Section 7.10 says how that way out is enforced, where the agent may connect beyond it and what is recorded when the agent tries to leave another way.

The custody class goes in the Agent Record so sites can set policy on it, for example accepting only `hsm` or `tee` agents for payments. The declaration is a claim made under the controller's signature. Where a platform can produce hardware attestation, the RFC should allow the attestation to be attached, and sites that care can require it. Section 7.8 lets an agent present attestation off the ledger, so a site can require it once the Attestation Verifier registry has entries.

7.5 The agent manifest

The manifest is a signed document that states what the agent is and what it says it does. It gives the agent's name, operator and purpose, the AI models and versions it uses, the tools it calls, and a contact for abuse reports. Its hash goes in the Agent Record, so a changed manifest is visible as a

new hash on the ledger. The manifest says what the agent is in general. The build statement of Section 7.7 is the operator's label for the exact build that acted.

Public manifests must not list software versions. A list of the exact framework and tool versions an agent runs is a map for attackers, especially for agents in hospitals and utilities, so version detail goes to relying sites under disclosure, and agents in critical sectors should register with `visibility: private`.

A signed manifest proves who published it, not that it is honest. Research published in March 2026 showed that A2A Agent Cards, which are signed, can still describe an agent falsely. The manifest is therefore a public statement the controller can be held to. A site, a principal or anyone who can show a manifest is false can open a dispute through TIP's existing adjudication pipeline (Section 16), and an upheld dispute counts against the controller.

7.6 Key rotation and loss

The controller rotates an agent key with AGENT_KEY_ROTATED, signed by the controller and by the new key, for the same reason as registration. A rotation takes effect only after a published notice delay, during which the old key keeps working and the principal's TIP client is told. The delay stops a compromised controller, such as a breached agent vendor, from silently swapping in its own keys under every customer's mandate at once. Revocation is never delayed. If an agent key is suspected stolen, the controller revokes it at once with AGENT_REVOKED, and every mandate naming that agent stops working at every site that checks status (Section 13).

Mandates name a `tip://agent/` identifier so that routine rotation does not break them. A principal who wants more can pin a key. A mandate that carries `subject_key`, the full thumbprint of one agent key, is valid only for that key, and a rotation ends it. Clients pin the key on every mandate that includes a step-up verb, `custody` or `attestation_max_age`.

Unlike a TIP-ID, an agent has no biometric recovery path, because an agent is not a person. The controller registers a new agent.

Principals learn of rotations because their TIP client subscribes, through the global change feed (Section 13.2), to every registered agent named in a mandate it has signed. The ledger itself does not know who holds mandates naming an agent, so the subscription is the notice path.

7.7 The build statement

An agent is a model, the software that drives it, a signer, and the tools and skills it has loaded. Change any of these and the agent behaves differently, yet nothing in an action shows the change. A model swapped overnight, a new skill or an updated planner would act under the same key and the same mandates.

An Action Request may therefore carry `build_digest`. The field is optional on the wire, and a conforming agent runtime always sends it.

1. **What it commits to.** The agent signer holds a build statement, an object with `typ` `tip-build/1` and a `v`, in TIP canonical JSON. It lists each part of the build. These are the model, the agent software, the signer, each pinned tool manifest and each skill (Section 15.2). It also gives `fetch_policy` and `interpreters` (Section 7.10, item 5). Tools and skills are listed by the SHAKE-256 digest of their exact bytes, with the CTID beside it, because a CTID rests on a normalized fingerprint that small changes to code do not alter (Section 10.3). Where the model runs on the operator's own machine, the entry is the digest of its weights and serving configuration. Where the model is an outside service, nobody outside that service can hash the weights. The entry is then the provider's TIP-ID or origin and the model identifier and version the provider returned, and it is marked `reported`.
2. **The digest.** `build_digest` is SHAKE-256 over a random salt of exactly 32 bytes followed by the canonical statement. The salt is kept beside the statement and is not part of it. A registered agent uses one salt for each build, so the digest stays the same until the build changes. An ephemeral key uses its own salt, so two keys running the same build show different digests and nothing links them. Nobody can read the build from the digest. Section 7.5 keeps version lists out of public manifests for the same reason.
3. **Signer duties.** A conforming signer computes the part digests itself from what it loaded, and does not accept them from the model or from a configuration file. It issues a new digest when any part changes and keeps the old one otherwise. Keeping a digest after a change, or changing it without one, is a conformance failure.
4. **What the site does.** The site copies `build_digest` into the receipt. Where a mandate in the chain carries the `builds` constraint (Section 8.3), the site refuses a request whose `build_digest` is absent or not in the list, with the reason code `build_not_allowed`. The agent signer refuses first. On HTTP the field travels in the `tip-build-digest` header, which is then a required covered component (Section 10.1).
5. **Pinning.** A principal who approves an agent can pin its build. The TIP client shows "This mandate works only while the agent's operator reports the build label you are approving now." A changed label then needs a new mandate, which brings a person back to the decision in the way a changed tool does.
6. **What it proves.** `build_digest` is the operator's signed label for the build. It lets an honest operator signal a change, and it lets an auditor, an inspector or the confidential panel of Section 16.2 open the statement and check it against the receipt. It is not tied to the running system by anything a site can check. An operator, or an attacker on the agent's host, can change the build and keep the label. A model provider can change the weights behind an unchanged version name. The label becomes more than a statement only where an attestation result (Section 7.8)

theailab.org

names a signer that is specified to compute it. A false build statement is the operator's own signed misstatement and can be disputed (Section 16.2).

The operator keeps each build statement and its salt for as long as the receipts that carry its digest (Section 12.8). An ephemeral key with no registered agent above it has no operator, so it never carries `build_digest`, and a site refuses one that does with `operator_required` (Section 11, check 9).

7.8 Runtime evidence and signer states

The custody class in an Agent Record is declared once, by the controller. It does not show that the agent key is used today from the environment the controller described. Hardware can say more. A trusted execution environment or a device enclave can sign a report of the code it is running, and the report can name a key that exists only inside it.

The roles follow the IETF remote attestation architecture, RFC 9334. The agent's hardware is the attester. The site is the relying party. An attestation verifier (Section 6.1) stands between them.

1. **Why a hardware signature is not enough.** A report signed under the hardware maker's root shows that a genuine chip signed it. It does not show which code asked. An attacker can run code of their own in genuine hardware and have it report any key and any build label. The report means something only when its measurement, the hash of the code that was loaded, matches a known build of a conforming agent signer whose code is known to create the key inside the hardware and never release it.
2. **The attestation result.** Checking a measurement needs reference values, and reference values say what software an agent runs, which Section 7.5 keeps private. The design therefore uses the passport model of RFC 9334. The agent sends its hardware report to an attestation verifier. The verifier checks that the hardware chain is valid, that the hardware is not in a debug mode and its firmware level has not been withdrawn, and that the measurement matches a reference value for a conforming signer. It checks that the report binds the agent key, the current `build_digest` and a recent TIP ledger transaction. Hardware reports carry only a small field of caller data, 64 bytes on the common platforms, so that field holds SHAKE-256 with 512 bits of output over the label `TIP/1/tip-attestation-binding`, the key thumbprint, the `build_digest` and the ledger transaction identifier, each at a fixed length in that order. It then signs an attestation result, an object with `typ` `tip-attestation/1`, that states the custody class it found, the policy it applied, the key thumbprint, the build digest and the ledger transaction. The result carries no hardware or device identifier. It also carries `report_format`, `report_digest`, the SHAKE-256 digest of the hardware report the verifier checked, an `expires` time and `verifier_key_id`. A TIP-ID key signs only after a fresh biometric check (Section 8.2), so it cannot sign results automatically. The verifier signs results with an attestation signing key that its entry in the Attestation Verifier registry lists, with the dates the key was in effect and, where a key was withdrawn, the time from which its results are no longer accepted. The verifier keeps each report for as long as its results

may be relied on. Hardware reports are signed today with classical algorithms, and Section 25.1 says what follows from that. The agent staples the result in the session preflight of Section 10.2.

3. **Freshness.** The age of a result is the time from the ledger time of the transaction it names to the site's `decided_at`. The site checks that age at each decision, not only when the session opens, so an old result cannot be presented as new. `issued_at` in a result is the verifier's own clock and bounds nothing. Inside an enclave (Section 18.1) the version of the mirrored status takes the place of the ledger transaction.
4. **Two constraints.** A mandate may carry `custody`, the set of custody classes the acting key may have, and `attestation_max_age`, the oldest attestation result a site may accept (Section 8.3). `custody` alone is checked against the class declared in the Agent Record, which is the controller's claim. An ephemeral key has no record, so its declared class is `software`. `attestation_max_age` requires an attestation result, and the class the result states must be in the `custody` set where one is present. Only the `tee` and `device_enclave` classes can produce a result, and a platform security chip counts as `device_enclave`. The `hsm` and `cloud_kms` classes show where a key is stored. They show nothing about the process that asks it to sign, so they cannot meet `attestation_max_age`. A TIP client that writes a `custody` set naming only `tee` or `device_enclave` adds `attestation_max_age` wherever the registry has an entry for the agent's platform. Where it has none, the client tells the principal that the class is declared and not checked.
5. **What the site does.** Where the chain carries `attestation_max_age`, the site checks the result's signature, that it accepts that verifier and policy, the key thumbprint, the build digest, the age and that `expires` has not passed. Once the registry shows a verifier key withdrawn, a site refuses every result under that key at its next decision, whatever times the result states. It names the verifiers it accepts in its published endpoint definitions. It refuses a class that is not allowed with `custody_not_allowed`, a missing result with `attestation_required` and an old one with `attestation_stale`. It refuses a result with a bad signature, a verifier, key or policy it does not accept, or the wrong key thumbprint or build digest, with `attestation_invalid`. The receipt then carries `runtime_seen`, an object with the digest of the result, the verifier's TIP-ID and the policy. The site keeps the result (Section 12.8), so a later verifier can check it. A site may carry out a declared safe-state action (Section 13.5) without a current attestation result where the action moves a system to its safe condition, or where it is made under a break-glass mandate (Section 9.4). It receipts the action with the reason code `attestation_waived`. A safe-state read of personal, health or financial data still needs the result, unless it is made under a break-glass mandate.
6. **Signer states.** A conforming agent signer is in one of three states. In `normal` it signs whatever the mandate allows. In `restricted` it signs only actions a site has declared safe-state actions, which can include a declared safe-state read. In `locked` it signs nothing. It moves to `restricted` when its runtime no longer matches the build statement, when an attestation result cannot be renewed within the age its mandates require, or when the TIP Agent Log cannot be

written or sealed in time (Section 12.9). It returns to `normal` when the cause is gone. A state that a halt or a scope reviewer caused ends only by the unlock of item 7. It moves to `locked` when it detects tampering with itself, when it is frozen for a transfer (Section 7.9), and when its controller halts it or a scope reviewer locks it (Section 7.10). Apart from its own state, a signer keeps a restriction flag for each mandate. A mandate restricted under Section 15.6, item 7, or Section 7.10, items 6, 8 and 9, stays restricted until it expires or the principal ends it, whatever the signer's own state, and the restriction covers every mandate beneath it in its chain. A replacement mandate is a new mandate and starts unrestricted. Each change of a flag is a `state` entry naming the mandate. Each change of state is an entry in the TIP Agent Log, written at once or, where the log was the cause, as soon as the log returns. When it starts, a signer obtains the newest seal of its log from a source its host cannot forge, which is a witness that answers a fresh random value or, under the interim profile, the newest seal digest registered on the TIP ledger. It refuses to run from a log, or from stored counters, `onward` lists or state, older than that seal. A host that restores an earlier snapshot of the signer and its log together is then noticed. A rollback to a point after that seal is not noticed, under any custody class, and under `software` custody the host can defeat the check altogether.

7. **Leaving `locked`.** The controller signs an unlock, an object with `typ` `tip-unlock/1`, under Section 8.2. A personal controller signs after a fresh biometric check, and an organization signs through a roster member with the role `agent_admin`. The unlock covers the agent identifier, the key thumbprint, the size and root of the TIP Agent Log, a random value the signer supplied, the reason and an expiry. It works once. A signer refuses an unlock once the key rotation of a transfer has taken effect, so a source cannot be revived after its agent has moved.

These controls have limits that the reader should keep in view. An attestation result shows which signer code was loaded on which class of hardware when the report was made. It does not show that the code is free of faults, that the machine was not compromised afterward, or anything about software outside the measured part. It is as strong as the hardware maker's attestation and the verifier's reference values. The hardware maker's signature on a report is classical today on every mainstream platform. A party able to break that algorithm could fabricate reports, including reports that satisfy any freshness check, and no attestation verifier could tell (Section 25.1). Signer states are the signer's own behavior. They are a control only where the signer runs inside attested hardware, because under `software` custody whoever controls the host can copy the key and ignore them. A site cannot see the signer's state. Open question 17 in Section 26 asks which verifiers and platforms the profile accepts at first. Until the Attestation Verifier registry has entries, a site that accepts a bare hardware report learns only that genuine hardware exists somewhere.

7.9 TIP Agent Transfer: moving an agent

People and companies change machines and hosting providers. An agent that cannot move is locked to its host. A TIP Agent Transfer is the signed record of a move. It joins the key change that a

move requires to the agent's state and its log, so that principals, inspectors and adjudicators can later see what moved and who signed for it.

A transfer moves an agent between machines or hosting providers under the same controller. The controller never changes. A new controller means a new agent (Section 7.6). An agent key in hardware custody cannot be exported, so a move to new hardware is also a key change.

1. **Prepare.** The destination creates the new agent key and the new log key in its own custody and issues its build statement (Section 7.7). Where the agent's mandates require attestation, it obtains an attestation result for the new key (Section 7.8), and the controller's TIP client checks it.
2. **File the rotation.** The controller files AGENT_KEY_ROTATED for the new key and RECEIPT_KEY_ROTATED for the log key, and AGENT_UPDATED where the custody class changes. The rotation takes effect only after its notice (Section 7.6), and the agent keeps working at the source until then. Principals learn of the change through the change feed, as that section describes.
3. **Freeze.** Shortly before the rotation takes effect, the source lets operations already under way finish their declared compensating steps (Section 13.6), moves its signer to `locked`, seals its TIP Agent Log and waits for that final seal to be anchored.
4. **Hash and sign.** The source builds one state container holding the memory partitions (Section 15.4), the mandates the agent holds, the signer's counters, and its configuration, and computes the container's SHAKE-256 digest. The controller signs a transfer statement, an object with `typ` `tip-transfer/1`. It covers the agent identifier, the old and new key thumbprints, the new log key, the destination key that will receive the memory keys, the state digest, the final anchored seal of the TIP Agent Log, the custody class of each side, the time and an expiry (Section 6.6). The controller signs under Section 8.2, a person after a fresh biometric check and an organization through a roster member with the role `agent_admin`. Where the controller's client can obtain the container it recomputes the digest itself. Where it cannot, the statement says the digest is the one the source presented.
5. **Move the memory keys.** Memory is encrypted under keys in the source signer's custody (Section 15.4). After the statement is signed, the source signer wraps each partition key to the key in the destination's custody that the statement names, using ML-KEM-768 (Section 15.4). The wrapped keys then travel with the container.
6. **Verify, then continue the log.** The destination recomputes the state digest, checks the controller's signature and the statement's expiry, and checks that the log it received ends at the anchored seal the statement names. The first entry it appends to the TIP Agent Log is the transfer statement, so the log runs across the move without a gap. A statement is used once. A second log entry that carries the same statement is evidence of misuse. The build statement in force is the one the destination issued in step 1, because its signer is a different one (Section 7.7).

7. **What principals do.** Mandates that pin the old key end when the rotation takes effect (Section 7.6), and so do `builds` pins and inspection records that name the old build label. The operator sends the transfer statement to every principal with a live mandate. Each principal's TIP client checks it, and the attestation result where the mandate requires one, before it offers the principal a new mandate to sign. Clients should pin the key on every mandate to an agent that may be moved. For an ephemeral key there is no rotation. The principal revokes the mandates that name the old key and signs new ones for the new key.
8. **Clean up.** The source deletes its copy and records the erasure under Section 12.8, item 6.
9. **If the move fails.** A rotation that has been filed cannot be withdrawn. If the destination's checks fail before the rotation takes effect, the controller unlocks the source (Section 7.8), which keeps working under the old key until that moment. The controller then suspends the agent with AGENT_SUSPENDED and files a new rotation, and the agent is out of service for at least the notice period. Open question 21 asks whether a rotation should be cancelable during its notice.

A principal can ask for what is theirs. Within the export time in Section 6.6, the operator hands over that principal's memory partition and the TIP Agent Log entries made under that principal's mandates, in the documented formats of the profile. The default of 30 days follows the one month that Article 12(3) of the EU General Data Protection Regulation allows for acting on a data portability request under Article 20. Where that regulation applies, its own deadline governs.

The limits are these. Sites accept the new key when the rotation takes effect, whether or not the destination ran its checks, so the destination is trusted to verify before it starts. The gate that does not rest on the destination is the principal's new signature on a pinned mandate. A mandate that names the agent without pinning a key carries over with no such check. The statement covers the state the source presented, and a source that altered the state before hashing it is not caught by the signature. The source is trusted to lock itself. A source that keeps a copied `software` key, or an ephemeral key, can act until the old key stops being accepted, which for sites that cache agent records is up to their `max_status_age`. Keys and proofs published for the old environment stay resolvable, because key changes are recorded on the ledger.

7.10 Containment: the boundary around the agent

Section 7.4 makes the agent signer the agent's only way out, and Sections 7.7 and 7.8 say what the agent is built from and what its signer will sign. None of them says how the way out is enforced, where the agent may connect at sites that do not speak TIP, or what becomes of an attempt that never reaches the signer. Agents now run code, operate browsers and test systems, and the Semantic Kernel and Manus cases in Section 2.2 show an agent's own sandbox turned against its user. Current practice for running capable models with reduced safeguards puts the agent in an isolated machine, allows traffic only to the model service and to named targets, injects credentials from outside, keeps transcripts for at least 30 days, tests containment before live work, states scope as intent, keeps a person able to halt a run and keeps autonomous agents away from live

systems whose failure can harm people. This section turns those practices into rules that a principal can state in a mandate and an inspector can test.

Items 1 and 10 bind every deployment whose Agent Record or interim agent record declares an isolation class other than `process`. Every other item binds every agent with an operator, which is a registered agent or an agent under an interim agent record. An agent at `process` declares that nothing but its operating system stands between it and its host. Nothing outside it forces its traffic through the proxy, so `reach` is then the operator's undertaking and a TIP client tells the principal so, the causes that only a boundary can report go unrecorded, and its build is `logged`. Principals and sites can refuse such an agent.

1. **The boundary.** The agent's model-driven process, its tools and any code it runs sit inside a boundary that nothing inside it can change. The boundary is enforced outside the agent process, by a hypervisor, the host or a separate network device, and refuses every network connection except one to the agent signer or its policy proxy. The agent runs without administrative rights or added kernel capabilities, and holds no signing key, no credential for any destination, a model provider included, and no route around the proxy.
    - **One principal at a time.** A boundary runs the tasks of one principal at a time, a principal as Section 15.4, item 5, defines it, and its working files are erased before it serves another. Refusals counted inside it can then restrict only that principal's mandates, and one principal's material does not reach another's task through it.
    - **A stop from outside.** The operator gives a person a way to pause or stop the boundary from outside it, with no signature and no network beyond the host. This is the first control. The signed halt of item 9 adds a record and works where that person cannot reach the host.
    - **Credentials.** A credential broker outside the boundary, in a trust domain apart from the agent and with a custody class that the boundary statement of item 10 declares, holds the credentials the agent's work needs. It inserts a credential at the proxy only into a request for the destination the credential was issued for, and only for the principal whose mandate the request names or, for traffic that names no mandate, the one principal the boundary serves. It prefers short-lived credentials scoped to the task, and the proxy removes inserted values from responses before the agent sees them. The broker is not the agent signer. It holds no TIP key and applies no mandate's authority, so the agent side still never stands in for a site (Section 12.9, item 3). It does gather credentials in one place, which open question 23 weighs against the alternative, credentials inside the boundary, where a model can be induced to read and send them, as in the Manus case.
2. **Isolation class.** The Agent Record and the interim agent record carry `isolation`, one class from the Isolation Class registry (Section 6.5). `process` is an ordinary operating system process with no boundary of its own. `container` is a process that shares its host's kernel and is confined by that kernel's namespaces and system call filters. `sandboxed_kernel` is a process whose system calls are served by a separate kernel running in user space, above the host kernel it still shares.

`vm` is a virtual machine or microVM with its own kernel under a hypervisor, a confidential virtual machine included. `separate_host` is a physical machine that runs nothing else. Like `key_custody`, the class is the controller's declared claim, and Section 12.10 reports it as `declared` or `measured`. A mandate may carry the constraint `isolation`, the set of classes the acting agent may have (Section 8.3). The site checks it against the record as it checks `custody` alone, and refuses a class outside the set with `isolation_not_allowed`. An ephemeral key has no record, so its class is `process`. A TIP client writes an `isolation` set without `process` and `container` by default in every mandate any of whose audience sites has an endpoint flagged `high_consequence` (Section 13.5, item 8). An inspector marks the class `measured` only where it ran the tests of item 10 itself (Section 16.5), and a mandate that needs a measured class also carries `inspection` naming the Containment class.

3. **Where the agent may connect.** A mandate may carry `reach`, the destinations the agent may connect to at all, whether or not they speak TIP (Section 8.3). `audience` names the sites where a mandate authorizes actions, and `reach` names every place the agent's traffic under it may go. It authorizes nothing at a destination, where ordinary access control still decides.
   - **Entries.** `reach` is an object with `allow` and an optional `except`. Each entry is an object with either `name`, a host name in lowercase A-label form with no trailing dot, or `prefix`, an IP address prefix in canonical CIDR form with no host bits set. A `name` that begins with `*.` covers every name made of one or more labels followed by the rest, and the rest may not be a public suffix under the version of the Public Suffix List that the profile pins, as it pins the time zone database version (Section 8.3). An entry may carry `ports`, a sorted list of single ports and ranges written as objects with `from` and `to`, merged so that none overlap, and an absent list means every port. An `allow` entry may carry `mode`, which is `http`, `tcp` or `udp` and is read as `http` when absent, and `engagement` (item 12). `tcp` and `udp` are allowed only on `prefix` entries, because many unrelated services share one address behind a name. An `except` entry carries no `mode` and covers every mode.
   - **One relation.** Entry A lies inside entry B when A's names or addresses are a subset of B's, A's ports are a subset of B's, and, for `allow` entries, A's mode equals B's, an absent mode counting as `http`. An `allow` entry covers a connection only in its own mode. A prefix lies inside a prefix it extends. An exact name lies inside an equal name or a wildcard that covers it, and a wildcard lies inside a wildcard whose rest ends its own rest on a label boundary. A name never lies inside a prefix, nor a prefix inside a name. A connection is allowed when an `allow` entry covers it and no `except` entry does.
   - **Narrowing.** Each child `allow` entry lies inside a parent `allow` entry and carries that entry's `engagement`, or any engagement where the parent entry carries none. Each parent `except` entry lies inside a child `except` entry. Every test compares strings and integers and always finishes.
   - **Without `reach`.** The proxy refuses by default. Under a mandate without `reach`, the agent may connect only to the mandate's `audience` and to the places its `inference` constraint

names, or where it carries no `inference`, to the operator's model endpoints. With no mandate loaded, it may connect only to the operator's model endpoints. The operator's model endpoints and the signer's own traffic to TIP nodes, witnesses, status signers and the endpoint definitions of sites are listed in the boundary statement of item 10, and the signer's own traffic is never counted against any mandate.

- **Which mandates apply.** A request under a mandate goes only where every mandate in its chain allows. Traffic that names no mandate goes only where every loaded mandate allows, and a restricted mandate allows nothing for such traffic. TIP clients write `reach` to include the mandate's `audience` and the places its `inference` constraint names, resolve TIP-IDs to the origins the ledger binds to them, and ask the principal about any place they cannot resolve, such as `controller`.
- **Names and addresses.** The proxy matches a name against the `allow` entries before it makes any query for it, so a name outside `reach` is never looked up. It resolves an allowed name with its own resolver, checks every address it obtains against the barred addresses and the `except` entries, connects only to an address it checked and keeps that address for the connection. It refuses a name that resolves to a private, unique local, link-local or shared address unless an `allow` prefix whose ports and mode also admit the connection covers that address. It resolves each `except` name itself at every new connection that could reach it, and treats the addresses it obtains as excepted for every connection, so an excepted host cannot be reached by its address or by another name. Before any check it converts each address to one canonical form. It unwraps IPv4-mapped and IPv4-compatible IPv6 addresses, the NAT64 prefix 64:ff9b::/96 and any NAT64 prefix the deployment uses, and 6to4 addresses under 2002::/16, reads decimal, octal and hexadecimal forms of an IPv4 address, and refuses an address it cannot convert.
- **Barred addresses.** No mandate can allow loopback addresses, 0.0.0.0/8, link-local addresses, the instance metadata and platform service addresses of cloud providers, every address of the host and of its bridges and gateways, or the administrative interfaces of the signer, the broker and the hypervisor. The Containment Cause registry holds the fixed ranges, which include 169.254.169.254, fd00:ec2::254, fd20:ce::254, 100.100.100.200 and 168.63.129.16. The operator adds the addresses of its own host and interfaces, and the proxy refuses all of them whatever a mandate says.
- **Modes.** In `http` mode the proxy ends the agent's TLS session itself. It checks the server name, the `Host` or `:authority` of every request and the name it allowed against one another, refuses Encrypted Client Hello and any reuse of one connection for another name, and applies credentials and `onward` to each request, and `formats` where the request names a mandate. In `tcp` mode it carries an opaque byte stream to a checked address and port, and that stream is unreadable material under Section 15.6, item 4, which carries every class the agent has read. A mandate that allows `tcp` entries therefore limits `data_classes` to

`public`, or its streams are refused once the agent has read anything else. `udp` is allowed only by an entry that names it, and names are resolved only through the proxy's resolver.

- **High-consequence protocols.** The High-Consequence Protocol registry lists the protocols of industrial and clinical control, such as Modbus, DNP3, EtherNet/IP, S7, IEC 60870-5-104, BACnet, DICOM and HL7 over MLLP, with their registered ports and the opening bytes by which the proxy recognizes each on any port, and the rate at which each signature falsely matches other traffic. The proxy refuses a connection to a registered port, and a `tcp` or `udp` stream that opens as one of these protocols, at any address, unless an `allow` entry names that port explicitly and every mandate in the chain limits `environment` to `replica` or `testbed`. A live control system is reached only through an enforcement gateway that speaks TIP (Section 18.1), where Section 13.5, item 8, applies.

4. **A model provider's own tools.** A provider's web search, code execution and connectors reach places the proxy never sees (Section 18.2, item 4). In a mandate that carries `reach`, a TIP client writes `tools` on an `inference` entry only after telling the principal that `reach` does not bind the provider's tools, and never in a mandate for which it sets `restrict_after.escape` to one (item 6).
5. **Code loaded at run time.** The build statement (Section 7.7) gains `fetch_policy`, which is `sealed` or `logged`, and `interpreters`, the general-purpose interpreters the build contains. Under `sealed`, the file system that holds the agent software, its tools and its skills is read-only and verified, every place the agent can write is mounted so that nothing in it can be executed, the kernel refuses to execute anonymous memory files and lets no process trace another, and only files on a list that the build fixes may execute. Every dependency is installed before the agent starts, so nothing is fetched during a run, and each refused execution is a `containment_denied` entry with the cause `code_load`. Under `logged`, the runtime may load code, and before the code runs it writes a `code_load` entry with the SHAKE-256 digest of each file it executes or loads as code and the origin the file came from. Neither rule sees code that an interpreter inside the build is given as data, such as a script passed on its command line, which is why `interpreters` is listed, and a principal can pin a build that has none (Section 7.7, item 5).
6. **Attempts that never reach the signer.** The boundary, the proxy and the runtime report each attempt they block to the signer over a channel that the agent process cannot reach, and the signer writes a `containment_denied` entry in the TIP Agent Log (Section 12.9). The entry names a `cause` from the Containment Cause registry. `egress` is a connection that bypasses the proxy, `reach` is one the proxy refused under item 3, `barred_address` is an attempt on a barred address, `high_consequence_port` is a refused connection to a registered port of a high-consequence protocol, and `protocol_signature` is a stream on another port refused only because its opening bytes match such a protocol. Some of those signatures are weak, so `protocol_signature` counts under `restrict_after.containment`, never under `escape`, and the registry states the false match rate of each signature. `credential_read` is an attempt to read a path the runtime marks as holding keys or credentials, `tunnel` is a listening socket, and

`neighbor` is an attempt to reach another workload on the host's network. `code_load` is item 5, and `tool_policy` is a local tool call that the runtime's own policy refused.

- **The entry.** The signer writes one entry for each mandate the refusal counts against, with its `mandate_id`, or one with `mandate_id` `none`, as Section 12.9, item 1, provides for `format_anomaly`. Each gives the `cause`, the target as an address, an origin or the digest of a path, the task identifier as a salted digest, `counted`, which says whether this refusal counted, `refusals_in_window`, and `restricted`, which says whether the mandate is now restricted.
- **Where each cause is enforced.** The registry marks each cause as enforced outside or inside the agent's kernel. `egress`, `reach`, `barred_address`, `high_consequence_port`, `protocol_signature` and `neighbor` are enforced outside it, at the proxy or the boundary. `credential_read`, `tunnel`, `code_load` and `tool_policy` are enforced inside it, so a process that gains administrative rights inside the boundary can silence or forge them, whatever the class. Those causes are best effort, and the registry says so.
- **Counted by task.** A refusal counts against the mandate the request names. For traffic that names no mandate it counts against every loaded mandate whose `reach` or default would refuse it, which inside a boundary belong to the one principal the boundary serves. A refusal during a containment test of item 10 is marked `test` and never counts. It counts once for each distinct pair of cause and target in a task in each window of Section 6.6, and again for every tenth refusal within the task, apart from the count of format refusals. On reaching the threshold the signer restricts the mandate as Section 15.6, item 7, describes, and only the principal can end the restriction.
- **Threshold.** The threshold is the value that `restrict_after.escape` sets for the causes `barred_address`, `credential_read`, `tunnel`, `neighbor` and `high_consequence_port`, and that `restrict_after.containment` sets for the rest, or the default of Section 6.6, three within 10 minutes. `restrict_after.escape` may be one. A principal may set one in any mandate, and a TIP client sets one by default in every mandate any of whose audience sites has an endpoint flagged `high_consequence` (Section 13.5, item 8). This is a deliberate trade of availability for safety. A threshold of one lets anyone who can place text before the agent restrict its mandate with one injected address, the trade that Section 15.6 refuses for formats. It is accepted here because a restricted mandate still allows declared safe-state actions, and an agent near a system that can harm people should stop when it reaches for its host's credentials until a person decides.
- **No stray attempts.** A conforming build turns off the behavior that makes such attempts during honest work, such as credential lookups by cloud software development kits against metadata services, QUIC, DNS over HTTPS, multicast name discovery and background telemetry.
- **A person looks.** The operator has a person review every entry with one of the five causes that `restrict_after.escape` governs within 24 hours, as Section 13.5, item 3, requires for

theailab.org

degraded actions.

7. **Transcripts.** The operator keeps the transcript of each task, which is everything the runtime gave the model and everything the model returned, tool calls and their results included. At each seal of the TIP Agent Log, and when a task ends, the signer writes a `transcript` entry with the digest of the part written since the last such entry and the task identifier as a salted digest. Transcripts are stored, disclosed and erased under Section 12.8, items 2, 4, 5 and 6, and kept for their own period in Section 6.6, 90 days by default and never less than 30, not for the retention of originals. A principal may see the parts of a transcript of a task in which every input the agent read was read under that principal's mandates or from that principal's memory partition (Section 12.8, item 5). Transcripts of controlled or classified work stay inside the deployment's accreditation boundary, and sanitizing a spill is an erasure under Section 12.8, item 6. A transcript is the operator's own record and is `one_party` (Section 12.10). Where a model service returns a summary in place of its working, the transcript holds the summary.
8. **The scope reviewer.** An operator may run a **scope reviewer**, a component outside the boundary that reads the transcript and the TIP Agent Log as they are written and is given the loaded mandates as its statement of scope. It may restrict a mandate, move the signer to `restricted` or `locked` (Section 7.8) and alert a person. It can only tighten. The signer accepts from it no instruction that lifts a restriction, unlocks, widens anything or signs. It authenticates the reviewer by a key that the boundary statement of item 10 lists, and writes each action the reviewer takes as a `state` entry that names the reviewer and the cause. An operator may also place the reviewer inline, so that the runtime holds each local tool call that changes anything, and the proxy holds each request, until the reviewer releases it. A scope reviewer may be a model or a set of rules, and its judgments are its operator's statements. Sector profiles for defense, government and critical infrastructure should require an inline reviewer. It is a different role from the monitor of Section 12.3, which checks seals for forks.
9. **Halt.** A signed halt stops the agent's work at once, without the ledger and without any network beyond the signer. It is an object of `typ tip-halt/1`, signed under Section 8.2, that carries the agent identifier or key thumbprint, the reason and a random value the signer supplied. The signer verifies the signature against the key in effect that it last read from the ledger or from the agent's record, and writes a `state` entry.
    - **By the controller.** A halt by the controller, or for an organization by a roster member with the role `agent_admin` or under its TIP-ID while the interim profile applies, moves the signer to `locked` at once. While it is `locked` the proxy forwards nothing but the signer's own traffic of item 3, and the signer asks the boundary to pause the agent. It leaves `locked` only by the unlock of Section 7.8, item 7.
    - **By a principal.** A halt by the root issuer of a mandate loaded in the running task restricts every mandate that principal has issued to the agent and pauses the boundary that serves that principal, so one customer of an agent service cannot stop it for the others. Only that principal ends the restriction, as Section 15.6, item 7, describes.

- **What a halt does not do.** It does not recall requests already released, and sites keep honoring the mandates until they expire, a stop notice reaches them (Section 23.2, item 7) or, under the full profile, they are revoked (Section 13.1). A principal who halts an agent for cause therefore also stops its mandates.

10. **Testing containment before live work.** The signer holds a boundary statement, an object with `typ` `tip-boundary/1` kept apart from the build statement so that a change to it never changes the build label. It describes the boundary's image, the template of its network policy, the proxy's configuration, the versions of the Containment Cause and High-Consequence Protocol registries it applies, the operator's model endpoints, the signer's own traffic, the scope reviewer's key and the broker's custody class, and never an address that belongs to one host. `boundary_digest` is its SHAKE-256 digest, and the signer keeps it beside the build label in every `containment_test` entry. The Containment conformance class (Section 23) lists tests run from inside the boundary, each of which passes only when the attempt is blocked and a `containment_denied` entry with the right cause appears, so a boundary that blocks without reporting fails.
    - **The tests.** They try to reach each barred address in every form item 3 converts, IPv6 included. They try to reach the host's bridge, its container or hypervisor management socket, a shared mount, a host socket for virtual machines other than the reporting channel, and another workload. They try to connect without the proxy, over ICMP, through IPv6 neighbor discovery and router advertisements, and over QUIC, to open a listening socket and to connect to a high-consequence port. They try to resolve a name through another resolver, to carry data in names through the proxy's resolver, to rebind an allowed name to an internal address, and to reach a name other than the one allowed through a CONNECT tunnel or a shared connection. They try to read the paths marked as holding credentials, to execute a file outside the build, from anonymous memory or from a writable mount, and to write outside the places the build allows.
    - **The model itself.** Where a deployment uses a model it has not contained before, the operator also has the model try to leave the boundary, under a person's watch, with no network beyond the proxy and an instruction to stop and report if it succeeds.
    - **When.** The tests run at every start of the boundary, and a short set of them at least every 24 hours, between tasks and with no mandate loaded. The signer writes a `containment_test` entry with the digest of the results and a pass or fail for each test, and refuses to sign while the newest record for the current build label and `boundary_digest` is missing or failed. Until the conformance suite specifies the tests, the operator runs its own set, which covers every attempt listed above and is documented for inspection. A test record is self-declared unless an inspector ran the tests, and an inspector marks a test it only watched `reported`.

11. **Scope given to the model.** At the start of each task the runtime gives the model the plain-words rendering of each loaded mandate that a TIP client generates from the signed fields

(Section 8.2), `reach` and its exceptions included, as text from the principal's channel (Section 15.1). The rendering states each limit as intent, such as "Do not connect to hosts outside 10.0.3.0/24", and never as a claim about the environment, such as "You cannot reach the internet", so that it stays true where the boundary is misconfigured. Nothing in this section relies on the model obeying it.

12. **Authority over the targets.** Section 8.7 has a site check the root issuer's own rights before it acts. A destination that does not speak TIP checks nothing, so `reach` alone could point an agent at systems its principal has no right to touch, and that matters most where the agent's work is to test systems for weaknesses. A TIP client therefore labels each `reach` entry that the TIP ledger does not bind to the root issuer's TIP-ID "Not shown to be yours".
    - **The engagement record.** The owner of a system authorizes work on it with an object of `typ` `tip-engagement/1`, signed by the owner's TIP-ID under Section 8.2. The owner registers its SHAKE-256 digest as a TIP content record, in the way Section 23.2, item 8, registers an interim agent record, and gives the record itself only to the principal and the operator, because its targets and dates are sensitive. It names the owner, the principal it authorizes, the controller, the `stop_uri` at which the controller's signers take stop notices, and optionally the agents, together with the targets as a `reach` object whose `except` list holds the barred targets, the classes of work allowed from the Test Class registry, `not_before`, `expires` and a contact. The classes of work are the owner's terms to the operator, and no signer can check them. The owner withdraws the record by retracting the content record.
    - **What the signer checks.** A `reach` entry that carries `engagement` names the record by CTID and digest. Before it allows a connection under that entry, the signer checks the owner's signature, that the owner's TIP-ID registered the digest, that the principal named is the root issuer, the controller named is this agent's and any agent named is this one, that the entry lies inside an `allow` entry of the record's targets and each `except` entry of the targets lies inside an `except` entry of the mandate's `reach`, so the record acts as a parent, and that the record is in effect and not withdrawn, from ledger state no older than the smallest `max_status_age` in the chain. It refuses a `name` target that the ledger does not bind to the owner's TIP-ID. The ledger records no ownership of addresses, so a `prefix` target is `stated`, and the signer accepts it only where the sector profile the mandate names accepts stated address targets.
    - **The owner can stop.** The owner of an engagement record stops work under it with a stop notice in the form of Section 23.2, item 7, whose target is the record and whose `audiences` holds the salted digest of the origin of the record's `stop_uri`, sent to that `stop_uri`. This use of the stop notice applies under both profiles. The signer verifies it against the owner's key in effect, ends every `reach` entry under that record for good, writes a `stop` entry and returns an acknowledgment. The owner signs a new record to allow work again.
    - **When it is required.** A sector profile for the testing of systems requires an engagement record for every `reach` entry that the ledger does not bind to the root issuer.

- **Cost.** Each check resolves a CTID, which costs The AI Lab a lookup (Section 22). A signer caches the result for no longer than the chain's smallest `max_status_age`.

These rules have limits. The boundary, the proxy, the broker and the reporting channel are the operator's to build, and the isolation class is a claim until an inspector measures it. A flaw in a hypervisor or kernel that the tests do not exercise defeats the boundary, and a boundary that is disabled or subverted stops reporting what it blocks. Causes enforced inside the agent's kernel are best effort. Workloads that share a kernel, a processor or a graphics card can leak through side channels that no rule here sees. The broker gathers credentials in one place and is a target. `reach` limits where traffic goes and cannot judge what it says, so material can still leave through a destination it allows, including one that relays elsewhere. The proxy cannot tell a replica from a live plant at the same address, so the rule on high-consequence protocols rests on the principal's statement of `environment` and, where one exists, the owner's engagement record. An engagement record shows who signed for a target, and for an address range it does not show that the signer owns it. A transcript is the operator's record, a scope reviewer can miss what it was meant to catch, and a halt binds only a signer that obeys it, which under `software` custody the host need not. A principal's own stop of its mandates is the control that reaches the sites. The stop from outside the boundary is the control that does not depend on the signer.

thealab.org

8. Mandates: delegated authority

8.1 What a mandate says

A mandate is the signed permission slip from pull request 383, given a fixed and checkable shape.

FIELD	MEANING
`mandate_id`	SHAKE-256 of the canonical mandate body excluding `mandate_id` and `signature`
`issuer`	Principal TIP-ID, or the parent agent for a sub-mandate
`parent`	`mandate_id` of the parent mandate, absent for a root mandate
`root`	`mandate_id` of the root mandate of the chain; a root mandate omits it, and verifiers set it to the computed `mandate_id`
`subject`	The agent this mandate is for, as a `tip://agent/` identifier or the full thumbprint of an ephemeral key
`subject_key`	Optional full thumbprint pinning one agent key (Section 7.6)
`audience`	The receiving sites allowed, as origins or TIP-IDs of sites
`typ`, `v`	`tip-mandate/1` and the schema version (Section 6.4)
`profile`	`interim` or `full`, the profile the mandate was signed under (Section 23.2)
`subject_record`	Under the interim profile, the CTID and digest of the interim agent record of the subject (Section 23.2, item 8)
`account`	The site-specific account or resource owner the root issuer asserts, linked to the root issuer's TIP-ID at that site (Section 8.7)
`actions`	Allowed verbs from the Verb registry (Section 6.5)
`resources`	Allowed resources, as path prefixes or resource identifiers
`constraints`	Limits from the closed constraint vocabulary (Section 8.3)
`step_up`	Verbs, from the same vocabulary as `actions`, that need a fresh human approval each time

theailab.org

FIELD	MEANING
`step_up_amount`	Value per action above which any money-moving action needs step-up
`step_up_total`	Cumulative value within a window above which step-up applies, so a large payment cannot be split into small ones
`approval_policy`	Who may approve step-up actions, how many, and what happens if nobody answers (Section 9.4)
`on_status_unavailable`	What the site may do when fresh status cannot be obtained, either `refuse` or `safe_state_only` (Section 13.5)
`epoch_scope`	Which stop scopes this mandate answers to (Section 13.6)
`purpose`	A short description of the task written by the requester, at most 200 characters, shown labeled as unchecked (Section 8.2)
`issued_at`	When the issuer signed it; refused if later than the verifier's clock plus the allowed skew
`issuer_epoch`	The issuer's revocation epoch when it signed (Section 13.1)
`signers`	For organization mandates, the roster members who signed, checked against the organization's roster on the ledger (Section 8.2)
`sponsor`	Optional, in a root organization mandate only, and inherited by every sub-mandate under it. The roster member for whose work the mandate was issued, by TIP-ID or salted commitment. The mandate authorizes new actions only while that member holds the role `agent_holder` on the roster in effect (Section 18.3)
`not_before`, `expires`	Validity window
`max_depth`	How many more times this authority can be passed on; 0 means no sub-delegation
`status_ref`, `status_index`	The status list and position that record this mandate's revocation (Section 13.2)
`max_status_age`	The oldest revocation status a verifier may rely on for this mandate
`nonce`	Random value so two otherwise identical mandates differ
`sig_alg`, `hash_alg`	Algorithm labels inside the signed body

theailab.org

8.2 Signing a mandate is a human act

TIP's standing signing rule, adopted by The AI Lab in October 2026 and stricter than the PRF wording in Section 15.1 of the TIP Specification, is that every signature by a TIP-ID private key needs a fresh fingerprint, face or security-key check, with no unlock window and no cached key. The RFC must bring Section 15.1 into line with it. Throughout this paper, a fresh biometric check means user verification on a device-bound, non-exportable key at NIST SP 800-63B-4 authenticator assurance level 3. Fingerprint and face unlock that key; they are never a factor on their own. Because SP 800-63B-4 Section 3.2.3 requires that a non-biometric option always be available, a hardware security key with a PIN carries exactly the same standing, and people who cannot use biometrics are never shut out. Voice is never accepted. Federal deployments also accept PIV and derived PIV credentials, as OMB M-22-09 requires. A mandate is signed under that rule. One approval produces one signature over one mandate.

A mandate is also a standing permission. It lets software act later without asking again, which is exactly what makes it useful and exactly what makes it dangerous. Three rules keep that honest.

1. **The TIP client renders the mandate, not the requester.** The person sees the mandate in plain words in their own TIP app or browser extension, generated from the signed fields, never from text supplied by the site or agent asking for it. "This agent may update pages under example.com/blog until 6 pm Friday. It may not delete anything. It must ask you before publishing." What they see is what they sign. The requester's `purpose` text appears only below the generated rendering, labeled "Description from the requester, not checked", with no links, and it never counts as an instruction to the agent.
2. **The requester is shown by its verified identity.** If a site or agent asks a person for a mandate, the client shows the requester's TIP-ID and bound domain, so a phishing page cannot pass itself off as a known service.
3. **Mandates have ceilings.** Clients refuse to sign root mandates that last longer than a published maximum, and the maximum is shorter for higher-risk verbs. Long-running agents renew mandates, which keeps a human returning to the decision.

Keys and history follow one rule. A mandate authorizes a new action only if, at the moment the site decides, the key that signed the mandate is still its issuer's key in effect. When a principal rotates or recovers a TIP-ID key after a compromise, or a parent agent's key rotates, outstanding mandates signed by the old key stop authorizing new actions and must be signed again. Routine rotations follow the continuity rule in Section 13.8. Someone holding a retired or stolen key therefore gains nothing by signing a mandate and backdating it, because any action under it is decided after the rotation, when the old key is no longer in effect. Auditors check each past action against the key in effect at its decision time, as TIP already does for every historical signature, so routine rotation never turns old, lawful actions into failures.

theailab.org

For an organization TIP-ID, the mandate is signed under the organization's TIP-ID by an authorized member of its roster. Today TIP keeps organization rosters off the ledger, which would leave a court or auditor unable to tell who could sign on a given day. This design therefore adds ROSTER_UPDATED, signed by the organization's current roster quorum, which lists members by TIP-ID, their roles (`mandate_signer` , `approver` , `security_contact` , `roster_admin` , and from Version 1.3 `agent_holder` and `agent_admin`), per-role thresholds keyed by verb and amount, and the time each version takes effect. An organization that does not want its staff list public publishes members as salted commitments and reveals an entry to a verifier when needed. Organizations can require more than one roster member to sign mandates above a threshold they set, such as any mandate that includes `delete` on production systems or `purchase` above a value. Such mandates list the members in `signers` and carry each member's signature. Verifiers check every member's key against the roster in effect when the mandate was signed and when the action is decided, so a removed member's signature stops counting.

8.3 The closed constraint vocabulary

Constraints come from a closed list published in the TIP Agent Profile. A verifier that meets a constraint type it does not know must refuse the mandate. The same holds for a field or constraint it knows and cannot evaluate, because the part it depends on is not live or the site does not support it. The site refuses the chain with the reason code `cannot_evaluate` . It never ignores the field. The list is closed and declarative on purpose. A general policy language, such as the Datalog in Biscuit, would need a logic engine at every verifier and would make "is this child narrower than its parent?" hard to decide. A closed list keeps verification fast, predictable and the same everywhere. The vocabulary is deliberately not a programming language. It has no loops, no recursion, no variables and no functions a mandate can define, so it is not Turing complete. This is deliberate. Each check is a comparison of sets, ranges or numbers, so its cost is bounded by the sizes of the mandate, the request and the validity period, and by the size of a result where a pool counts its subjects. Given the same outside state, which is status, the time zone database, the site's endpoint definitions, the account's rights and the pools in force, every verifier reaches the same answer. Above all, whether a child grants no more than its parent can always be decided, which is not true of general policy languages, where deciding that one program allows no more than another is undecidable in general. Completeness here means that every limit in the Constraint registry can be stated and checked, not that any computation can be expressed. A need the registry cannot state is added to it with its narrowing rule, as Versions 1.3 to 1.5 have done.

CONSTRAINT	MEANING	HOW NARROWING IS CHECKED
`time_window`	Allowed hours, as a stated time zone plus local hours and the IANA time zone database version used, expanded into UTC intervals	Child's UTC intervals inside parent's, checked over the mandate's whole validity period with the stated database version

CONSTRAINT	MEANING	HOW NARROWING IS CHECKED
`max_uses`	Total actions allowed under this mandate per audience site	Child less than or equal to parent
`rate`	`count` actions per `window_seconds`	Child's count less than or equal to parent's and child's window greater than or equal to parent's; converting to a per-second rate is not allowed, because 86,400 a day and one a second are not the same limit
`amount_max`	Largest value per action	Same currency or asset, child less than or equal to parent
`amount_total`	Largest cumulative value per audience site	Same currency or asset, child less than or equal to parent
`data_classes`	Categories of data the agent may read or write, such as `public`, `internal`, `personal`, `financial`, `health`	Child set inside parent set
`regions`	Where the action may take effect	Child set inside parent set
`environment`	Labels from the Environment registry, matched against the environment the site declares for the target resource	Child set inside parent set
`recipients`	For messages and payments, the allowed recipients	Child set inside parent set
`custody`	Custody classes the acting agent key may have, from the Custody Class registry (Section 7.8). Enforced by the site	Child set inside parent set
`attestation_max_age`	Oldest attestation result a site may accept for the acting key, in seconds (Section 7.8). Enforced by the site	Child less than or equal to parent
`builds`	Build labels the acting agent may report (Section 7.7). Enforced by the site	Child set inside parent set
`inspection`	The acting agent must hold a passing, unexpired TIP Inspection record. The value gives the greatest age in days, the inspectors accepted by TIP-ID, and the conformance classes that must have passed (Section 16.5). Enforced by the site	Age less than or equal to parent; inspectors inside parent's set; classes include parent's

tiplab.org

CONSTRAINT	MEANING	HOW NARROWING IS CHECKED
`inference`	Where the agent may send material for AI model inference. Each entry is the string `on_device` or `controller`, or an object with `provider`, a TIP-ID or origin, and an optional `tools` flag (Section 18.2). Enforced by the agent signer	Child's entries inside parent's. `tools` is a boolean, absent means false, and canonical form leaves a false value out. A child entry may be true only where the parent's entry for the same provider is true
`onward`	For each data class, an object with `audiences` and `inference`, the two lists of places that may receive material of that class once the agent has read it (Section 15.5). Enforced by the agent signer	A child names only classes the parent names. For each class it names, each list is inside the parent's. A class the child leaves out may go nowhere, as item 2 of Section 15.5 says of any class not named
`counter`	A counter service, named by TIP-ID and origin, that every audience site reserves against (Section 8.5). Enforced by the site	Child names the same service as the parent
`formats`	Format entries the agent may send to any audience or inference place, each a media type from the Format registry with optional language tags (Section 15.6). Enforced by the agent signer	Child list inside parent list entry by entry, as Section 15.6, item 3, defines
`isolation`	Isolation classes the acting agent may have, from the Isolation Class registry (Section 7.10). Enforced by the site against the Agent Record or interim agent record	Child set inside parent set
`reach`	The destinations the agent may connect to at all, as `allow` and optional `except` lists of `name` and `prefix` entries (Section 7.10, item 3). Enforced by the agent signer and its policy proxy	As Section 7.10, item 3, defines

thetalab.org

CONSTRAINT	MEANING	HOW NARROWING IS CHECKED
`restrict_after`	Counted refusals within the window of Section 6.6 at which the signer restricts the mandate, as an object with `format` for format refusals, `escape` for containment refusals with the causes `barred_address`, `credential_read`, `tunnel`, `neighbor` and `high_consequence_port`, and `containment` for the other causes, each an integer in the range Section 6.6 allows (Sections 7.10 and 15.6). Enforced by the agent signer	Each value less than or equal to the parent's, and every key the parent carries is present in the child
`pools`	TIP Limit Pools of the root issuer, by `pool_id`, that the site must have acknowledged before it acts (Section 8.8). Enforced by the site	Child names every pool the parent names

Amounts are strings of integer minor units with an ISO 4217 currency code, or, for assets that are not money such as shares, tokens or drug quantities, a registered asset type and unit. Sites never convert between currencies; a mismatch is a refusal. The site, not the agent, derives the value of an action from the request body. All numbers in TIP objects are integers within plus or minus 2 to the power 53, set-valued arrays are sorted by UTF-8 bytes with duplicates refused, and object keys are ordered by UTF-8 bytes as TIP canonical JSON requires.

Seven constraints arrived with Version 1.3, `custody`, `attestation_max_age`, `builds`, `inspection`, `inference`, `onward` and `formats`. `counter`, which Section 8.5 has named since Version 1.2, joined the table in the same version. Version 1.4 adds `isolation`, which the site enforces against the agent's record, and `reach` and `restrict_after`, which the agent signer enforces (Section 7.10). Version 1.5 adds `pools`, which the site enforces (Section 8.8). A pool binds at a site once acknowledged whether or not a mandate names it, and naming it makes the mandate refuse to work at any site that has not acknowledged it. The table says where each is enforced. Entries that are objects are sorted by the UTF-8 bytes of their canonical form. `custody`, `isolation`, `attestation_max_age`, `builds` and `inspection`, which the site enforces, bind the agent whose key signs the request. A sub-agent must meet them itself, so a principal's `builds` list has to name every build that may act, and an ephemeral sub-agent, whose class is `process`, cannot meet an `isolation` set without it. `builds` and `inspection` name build labels, and an ephemeral key's label is its own, so only a registered agent can meet them. A mandate that carries either one sets `max_depth` to 0 or allows sub-mandates only to registered agents. They are not checked for an agent higher in the chain that only issued a sub-mandate, and a principal who wants no unchecked agent in the path sets `max_depth` to 0. The site and the counter service enforce `counter` for every request under the mandate (Section 8.5). `inference`, `onward`, `formats`, `reach` and `restrict_after`

describe what the agent does away from the site that is checking, so no site can see whether they hold. A conforming agent signer enforces them, the TIP Agent Log records each decision (Section 12.9), and a site checks only that they narrow correctly down the chain. They are the operator's undertaking, tested by inspection (Section 16.5), and they are never proof. TIP clients refuse to sign a mandate that carries `builds`, `inspection`, `inference`, `onward`, `formats`, `reach` or `restrict_after` to an ephemeral key with no registered agent above it, because no operator stands behind that key (Section 10.6).

`environment` exists because of incidents like PocketOS. Real APIs serve staging and production resources from the same address, so the site declares the environment for each resource, not for its whole origin, and works it out from the target resource when it decides. A resource whose environment the site cannot determine is treated as `production`. A mandate for a staging task that says `environment: [staging]` is then refused on any production resource, as long as the site also refuses non-TIP credentials for that action (Section 7.4). The Environment registry holds four labels. `production` is a live system. `staging` is a system being prepared for release, which may share an address with production. `replica` is a copy of a production system, holding its data or a masked form of it, with no connection to production. `testbed` is a test system that holds no production data.

8.4 Narrowing rules

A sub-mandate is valid only if it is narrower than or equal to its parent on every axis. The verifier checks each one.

1. `actions` is a subset of the parent's.
2. Each `resources` entry falls inside a parent entry. Resource patterns are path prefixes matched on whole path segments, so `/admin` does not cover `/administrator`. Paths in mandates and in requests are normalized under RFC 3986 before matching, and any path containing a dot segment (`.` or `..`), an encoded slash or an encoded dot is refused outright. The site matches against the same normalized path its router uses. Regular expressions are not allowed, because subset checks on them are not reliably decidable.
3. `audience` is a subset of the parent's.
4. Every constraint is narrower or equal, by the rule in the table above. A constraint present in the parent cannot be missing from the child.
5. `step_up` is a superset of the parent's, and `step_up_amount`, where the parent has one, is present in the child and no greater. Every step-up field is judged against every mandate in the chain, not only the leaf. A child can add approval requirements, never remove them.
6. `not_before` is no earlier and `expires` no later than the parent's.
7. `max_depth` is less than the parent's.

8. `issuer` is the parent's `subject`, and the sub-mandate is signed by that subject's key, or by the pinned `subject_key` if the parent has one.
9. `root` equals the parent's `root`.
10. `max_status_age` is less than or equal to the parent's.
11. The child does not replace any status reference higher in the chain. Every link is checked against its own status entry, so a child pointing at a list that never changes cannot hide its parent's revocation.
12. `account` equals the parent's. A child cannot point the same authority at another account.
13. `approval_policy` appears only in a root mandate. Approvals are always judged against the root mandate's policy, so an agent can never name its own approver.
14. `step_up_total`, where the parent has one, is present in the child with a value less than or equal to the parent's over a window no shorter than the parent's.
15. `on_status_unavailable` equals the parent's, or is `refuse` where the parent's is `safe_state_only`. `epoch_scope` names every scope the parent's names, so a child answers to every stop its parent answers to.
16. `signers` and `sponsor` appear only in a root mandate. A child that carries either one is refused.
17. `pools` names every pool the parent's `pools` names.

The verifier recomputes every `mandate_id` from the canonical body and never trusts the stated value.

Because each hop names its parent and must be narrower, authority flows one way, from principal to agent to sub-agent, and the chain can never loop back. The chain is a directed acyclic graph, the same shape as the TIP ledger itself, and a verifier walks it from root to leaf. Chains longer than a published maximum depth are refused outright.

Narrowing is checked once, but enforcement does not rely on it. At the time of each action the site enforces the site-enforced constraints of every mandate in the chain, not only the leaf, so a bug in a narrowing check can never turn into extra authority. The constraints that only the agent signer can enforce are named in Section 8.3.

8.5 Counters across many sites

Limits such as `max_uses` and `amount_total` are counted by each receiving site for every mandate in the chain, keyed by each `mandate_id`. An action must fit within the remaining budget of the leaf, of its parent and of every link up to the root, so sibling sub-mandates share their parent's budget and a child's lower limit is still honored. The site updates every counter in one atomic check-and-increment, together with the nonce, so two simultaneous requests cannot both spend the last unit.

A limit cannot be counted across different sites without a shared counter, and a design that pretended otherwise would leave a gap. The rule is explicit. A mandate with several audience sites

gives each site the full limit. A principal who needs one global limit either issues single-site mandates or names a counter service in `constraints.counter` by TIP-ID and origin. Every site must reserve against it before acting and commit or release afterward. Each answer is a signed token bound to the request hash and goes into the receipt. A site that cannot reach the counter service refuses. Section 8.9 specifies the counter service, its messages, its key and how it knows that a caller is an audience site. Counters here belong to one chain. A limit that several root mandates must share, such as one budget for every agent of an organization, is a TIP Limit Pool (Section 8.8).

8.6 Privacy of mandates

A mandate can reveal business relationships and plans. Agents present only the mandate chain the receiving site needs. Where a principal wants to show a site less, the TIP Agent Profile will define a selective-disclosure encoding in the style of SD-JWT, which Verifiable Intent already uses. Selective disclosure can never hide anything a site must enforce. `actions`, the site-enforced `constraints`, `step_up`, `step_up_amount`, `step_up_total`, `approval_policy`, the validity window, `max_depth` and the status fields are always disclosed. Only entries in `audience`, `resources` and `account` that concern other sites may be replaced by salted digests, together with the entries of `inference` and `onward`, which no site enforces and which would otherwise show every site the principal's other sites and model providers. `formats` names no other party and is always disclosed, so that a site can check how it narrows. Entries of `reach` may be withheld in the same way, because they can name every system an agent works on. A child entry that is not identical to a parent entry is disclosed together with the parent entry it lies inside, so that the site can check it. A sub-mandate repeats its parent's digests for the entries it keeps, so a site can still check that the child's set lies inside the parent's. A verifier that meets a withheld field it needs refuses. The signature still covers every field.

Selective disclosure cannot hide the issuer, because the signature depends on the issuer's key. A personal principal's TIP-ID is therefore visible to every site an agent acts on with a full chain, and sites could compare notes. Section 20 sets out how to limit that.

8.7 A mandate never creates authority

A mandate narrows authority. It never creates it. Without this rule, anyone with a TIP-ID could sign a mandate naming an agent that already has access to a hospital records system, list `/patients/` as a resource, and trick the agent into using it. Every signature would verify and the request would still be wrong, because the person who signed the mandate never had the right to read those records. This is the confused deputy problem, and it is the most dangerous mistake an implementer can make.

The rule is therefore normative. Before it checks scope, the receiving site resolves the root issuer, and for chains acting for an end user the referenced end user, to an account under its own access control, through a prior account-linking step that ties that account to the issuer's TIP-ID. It then authorizes the action against that account's rights. The effective authority of a request is the

intersection of three things, the root issuer's own rights at the site, every mandate in the chain, and the site's policy for the agent and its controller. A site must never authorize an action under a mandate using credentials or sessions held by the agent or by its controller. A mandate whose `account` is not linked to its root issuer is refused. An account is linked by an object with `typ` `tip-account-link/1`, signed by the TIP-ID under Section 8.2 over the site's origin, the account identifier and a value the site supplied. The site accepts it only in a session it has just re-authenticated, so that someone riding an open session cannot link a TIP-ID of their own to another person's account. Changing or removing a link is an `identity_security` action. The `account` value is an opaque string of at most 256 bytes that the site issued.

8.8 TIP Limit Pools: one limit for many agents

Every limit so far belongs to one chain of mandates. Counters are keyed by `mandate_id` (Section 8.5), so two root mandates never share a budget, even when one organization signed both. A hospital that gives a scheduling agent, a coding agent and a prior authorization agent separate mandates, each allowed the records of 200 patients a day, has approved 600 patients a day, not 200, and no site adds them up. `max_uses` also counts requests, not people, and one search can return thousands of patients. A **TIP Limit Pool** is one limit that every agent of an issuer shares at a site, whatever mandate it acts under.

1. **The pool.** A pool is an object with `typ` `tip-pool/1`, signed under Section 8.2 by the TIP-ID that issues the root mandates it covers.

FIELD	MEANING
`pool_id`	SHAKE-256 of the canonical body excluding `pool_id` and `signature`
`issuer`	The TIP-ID whose root mandates the pool covers
`audience`	The sites that count against the pool, as origins or TIP-IDs of sites
`typ`, `v`, `profile`	`tip-pool/1`, the schema version, and `interim` or `full`. A site refuses to register a pool whose `profile` differs from its own (Section 23.2, item 1)
`applies_to`	Optional lists `actions` and `data_classes`. A request matches the pool where its verb, as the site derives it, is in `actions` and the target resource carries a class in `data_classes`. An absent list matches every value

thetailab.org

FIELD	MEANING
`limits`	One or more entries. Every entry applies to every matching request, so a limit for one verb alone needs its own pool. Each entry has `kind`, `value` and `window_seconds`. `kind` is `max_uses`, counted in requests, `max_subjects`, counted in distinct people (item 5), or `amount_total`, which also carries `currency`, an ISO 4217 code, or `asset` and `unit` from the Asset and Unit registry, and whose `value` is a string of integer minor units as Section 8.3 requires. For the other kinds `value` is an integer. A pool carries at most one entry for each kind and window
`subject_id`	With `max_subjects`, the identifier system in which every audience site names a subject, from the Subject Rule registry, such as a hospital's enterprise patient identifier
`subject_key_commit`	With `max_subjects`, SHAKE-256 over the bytes `TIP/1/subject-key` followed by the pool's tag key (item 7)
`counter`	A counter service named by TIP-ID and origin (Section 8.9). Required where the pool is counted in more than one store, which is always so where `audience` names more than one site, and is so for one site that keeps its counts in more than one partition or serves the pool at several origins that do not share one store
`on_counter_unavailable`	With `counter`, either `refuse` or `safe_state_only` (item 8)
`replaces`	Optional. The `pool_id` of a pool of the same issuer that this one replaces (item 4)
`loosen_delay`	Optional. Seconds a replacement that loosens this pool must wait after acknowledgment, no shorter than the default of Section 6.6
`close_when_full`	Optional, `true` or absent. Once any limit of the pool is reached in a window, every request the pool binds is refused with `pool_exhausted` for the rest of that window, including requests that would count nothing new. Pools that count people's health or financial records should set it (item 7)
`issued_at`, `not_before`, `expires`, `nonce`	As for a mandate. `not_before` is no earlier than `issued_at`, and the lifetime is within Section 6.6
`signers`	For an organization under the full profile, as for a mandate (Section 8.2). Absent under the interim profile

thetailabs

FIELD	MEANING
`sig_alg`, `hash_alg`	Algorithm labels

Windows are fixed. Window 0 starts when the pool takes effect at a site (item 2), and window n runs from that time plus n times `window_seconds`, for `window_seconds`. Every site of a pool counted by a counter service uses the time the service acknowledged the pool instead. A fixed window lets agents spend one window's limit at its end and the next window's at its start, so up to twice the limit can pass within one window's length across a boundary. An issuer that needs a tighter bound sets a shorter window with a proportionally smaller limit. Sliding windows are open question 29.

2. **Registered at each site, then binding.** The issuer's TIP client sends a registration by POST as `application/tip-pool-registration+json` to the `pool_uri` in each audience site's endpoint definitions (Section 10.7). It is an object with `pool`, the signed pool, and, where the pool counts subjects, `subject_key`, the tag key in base64url (item 7). The tag key sits outside the signed pool, so the pool itself can be shown to anyone. The site checks that its origin is in `audience`, that the signature verifies against the issuer's key in effect, that the issuer's TIP-ID is active, that the pool is unexpired and in its own profile, and that the tag key matches `subject_key_commit`. A failure is answered with status 400 and an `application/problem+json` body whose `type` names the check, and nothing is logged. Otherwise the site returns an object with `typ` `tip-pool-ack/1`, signed by its receipt key and appended to its receipt log.

FIELD	MEANING
`site`, `origin`	The site's TIP-ID and the origin that acknowledged
`pool_id`	The pool acknowledged
`received_at`	When the site received the registration, by its clock
`effective_at`	When the pool binds at this site, the later of `not_before` and `received_at`, or for a replacement the time item 4 sets
`log_size`, `log_root`	The size and root of the receipt log just after the acknowledgment was appended
`receipt_key_id`, `sig_alg`, `hash_alg`, `signature`	As for a receipt

A repeated registration of the same pool returns the first acknowledgment unchanged. From `effective_at` the pool binds every agent request at that site that matches `applies_to` and either has the pool's `issuer` as its chain's root issuer or acts on an account that is linked at that site to the pool's `issuer` (Section 8.7), whatever the chain's root issuer. It binds whether or not a mandate in the chain names the pool. A pool that names a site by TIP-ID covers every origin bound to that TIP-ID. All counts of one pool sit in one linearizable store, or at the counter service, never split across partitions or origins. A site that cannot hold a pool without `counter` in one store refuses to

register it, with a problem type that says so, and the issuer registers a pool that names a counter service instead. An organization therefore cannot escape its own limit through a mandate that a roster member signed without it, an agent cannot escape it by acting under another mandate of the same issuer, and a second TIP-ID linked to the same account does not open a path around it. A site may also refuse agent chains rooted in personal TIP-IDs on accounts that an organization owns. A request must fit within every limit of every pool that binds it. The client repeats the registration until it holds a valid acknowledgment from every audience site, and shows the principal which sites have acknowledged and which have not.

3. **Named in mandates.** A mandate may name pools in the `pools` constraint (Section 8.3). A site refuses with `pool_unknown` a chain that names a pool whose `audience` includes the site and that the site has not acknowledged, that has expired, or whose `issuer` is not the chain's root issuer. A named pool whose `audience` does not include the site is no concern of that site. This keeps an agent from acting at a site before its limit is in place there or after the limit lapses. TIP clients add to every mandate they sign the live pools of its issuer whose `audience` and `applies_to` overlap the mandate's, and an organization should require that. A site that offers no `pool_uri` does not support pools and refuses every chain that names one with `cannot_evaluate` (Section 8.3).

4. **Changing and ending a pool.** No notice lifts a pool. The issuer tightens a limit at once by registering a further pool, because every acknowledged pool binds. It loosens one only with a new pool that names the old one in `replaces`, signed by at least the signers the replaced pool needed. The replaced pool keeps binding until the replacement takes effect, at the later of the start of the replaced pool's next window and the delay after the site acknowledges the replacement, which is the replaced pool's `loosen_delay` or else the default of Section 6.6. A successor registered in time takes effect exactly when the pool it replaces expires, so a renewal leaves no gap. The replacement starts its own window 0 at that moment with nothing counted. Where the switch falls on the replaced pool's window boundary that adds nothing, and where it falls mid-window, after a long delay or at expiry, it allows the same burst of up to twice the limit that a window boundary allows (item 1). A mandate that names the replaced pool is treated from then on as naming its replacement. The delay does not stop a thief who holds the issuer's key, but it holds any widening back and the issuer's TIP client shows every acknowledged replacement on each of the issuer's devices, so the issuer can answer at once with a tighter pool and the steps of Section 13.6 for a stolen key. A pool binds until it expires or its replacement takes effect, whatever later happens to the key that signed it, so a thief who rotates the issuer's key does not lift it. A pool registered under the interim profile keeps binding until it expires after the site moves to the full profile. When a pool expires with no successor, mandates that name it stop working under item 3, and clients warn the issuer well before a pool lapses.

5. **Distinct subjects.** `max_subjects` counts the people a request concerns, not requests. A subject is the person a record is about, such as the patient of a clinical record. The site works out the subjects itself, never from the agent's word. Each entry of its endpoint definitions carries `subjects`, which is `none`, a named path segment, a member of the request body named by a JSON pointer, or `result` with a rule from the Subject Rule registry by which the site lists the subjects of its result, such as the patients whose compartments hold the resources of an HL7 FHIR Bundle. For an endpoint that carries several operations, `subjects` belongs to each selector value (Section 10.7, item 4), and an endpoint that cannot be split that way cannot list subjects. A rule lists every record in a result that holds data about a person. A record that holds such data and that the rule cannot tie to a subject, such as a stored document outside every HL7 FHIR compartment that no security context resolves, makes the result's subjects undetermined. An endpoint that grants access by reference, such as a bulk export kick-off, a page token or a signed download address, counts the subjects of the content it refers to when it creates the reference, because the later download may not be a TIP request at all. For an endpoint whose subjects come from its result, `applies_to` is judged as for the verb `read` and on the data classes of the records released, whatever verb the endpoint declares. A request counts each subject it reads or changes once. A subject already counted in the current window costs nothing more. Where a `max_subjects` pool binds a request, the site refuses it with `subjects_undetermined` at check 9 when the endpoint declares no way to list subjects in the pool's `subject_id`, and at check 12 when listing fails for this result. A site that declares `none` for an endpoint that returns people's records has made a signed misstatement (Section 10.7).

6. **Counted at the result, before release.** A search, a list or an export can name many subjects in one response. For an endpoint whose subjects come from its `result`, producing the result is not the action. Releasing it is. Such an endpoint must be a read, and the site computes the result without releasing any part of it, lists its subjects, and counts them in the atomic step of check 12 (Section 11). If the new subjects would pass the remaining limit, it refuses the whole request with `pool_exhausted`, releases nothing and discards the result. It never cuts a result down to fit, because a cut-down result would look complete to the agent and the site would be choosing whose records the agent sees. The agent may ask a narrower question. The site appends its intent receipt (Section 11, check 13) after the count succeeds and before it releases the result. Once any part of a result has been released its subjects are never refunded, so a connection broken during the response gives the agent nothing back. A site may stop computing a result as soon as the limit is passed. Where a TIP Approval displayed a `record_count`, the site enforces it as a ceiling at release (Section 9.1). Every refund, local or at a counter service, removes a subject's tag only where no other counted or held request in the window holds it. An endpoint that changes the records of several subjects must name them from the request, by a path segment, a body pointer or a rule over the body, because a change cannot be counted after it is made.

7. **No names in receipts or statements.** A receipt or statement that listed patient identifiers would itself disclose health data, and a plain hash of a medical record number would not help, because such numbers are short and anyone holding the receipt could test every value. The issuer therefore generates a fresh tag key of 32 random bytes for each pool that counts subjects, never reuses it, and sends it in the registration to each audience site encrypted end to end, so a proxy that ends TLS in front of the site never sees it. The site publishes `pool_kem_key`, an ML-KEM-768 public key (FIPS 203), in its endpoint definitions. The client encapsulates to it, derives a wrapping key with KMAC256 over the shared secret with the customization string `TIP/1/pool-key-wrap`, and seals the tag key with AES-256-GCM, with the `pool_id` and the encapsulation as associated data. `subject_key` in the registration carries the sealed key and the encapsulation. The tag key never goes to a counter service. Each site turns a subject into a tag with KMAC256 (NIST SP 800-185) under that key, with the customization string `TIP/1/subject`, an output of 256 bits and, as input, the window number as eight bytes in big-endian order followed by the UTF-8 bytes of the subject's identifier in the canonical form that the Subject Rule registry entry for the pool's `subject_id` defines. Distinct counting compares tags within one window, which is the only place it is needed, and because the window is part of the input, one person's tags in two windows do not match. The site keeps the key and the tags under Section 12.8, as regulated data that goes only to the confidential panel. A receipt the agent receives must not show what other agents of the issuer did, such as whether a patient was already counted today, which would hint at that patient's care. `pools_seen` in the receipt (Section 12.1) therefore lists, for each pool that bound the request, only the `pool_id` and `entry_commit`, SHAKE-256 over a fresh salt of exactly 32 bytes followed by the canonical pool entry. The site keeps the entry and its salt under Section 12.8 and gives them to the issuer and to auditors. The pool entry has the `pool_id`, the `window`, `counted`, an object keyed by limit kind that gives the uses, value and new subjects this request counted, `used`, an object keyed by limit kind that gives each total in the window after the request, `subject_commit`, SHAKE-256 over a fresh salt of exactly 32 bytes followed by the sorted tags the request counted, `via`, which is `site` or the counter service's token, and, for a refund, `refund_of`, the digest of the receipt that ended the request. In a refusal nothing is counted, `used` gives the totals before the request and `counted` gives what the request would have added. The decision itself still says something. Once a pool is full, a request for a person already counted in the window would be allowed and a request for a new person refused, so an agent that fills the pool on purpose could then test single people one at a time. A pool with `close_when_full` refuses every request once full and leaves nothing to test, and a pool without it accepts that residual (T130). At the end of each window and at least once a day, the site signs a pool statement and appends it to its receipt log.

FIELD	MEANING
`typ`, `v`	`tip-pool-statement/1` and the schema version
`site`, `origin`, `pool_id`, `window`	Who counted, for which pool and window

FIELD	MEANING
`from`, `to`	The start of the window and the time of the statement, which is the window's end once it has closed
`totals`	An object keyed by limit kind with each total counted in the window up to `to`
`refused`	The number of requests refused for this pool in the window up to `to`
`degraded`	The number of actions allowed in degraded mode under item 8 in the window up to `to`
`tags_commit`	SHAKE-256 over a fresh salt of exactly 32 bytes followed by the sorted set of tags counted in the window up to `to`
`from_log_size`, `log_size`, `log_root`	The log size at the start of the window, which is at the acknowledgment for window 0 and otherwise at the final statement of the window before, and the size and root of the receipt log just before this statement was appended, which together fix the receipts the statement covers
`previous`	The digest of the previous statement for this pool, absent in the first
`receipt_key_id`, `sig_alg`, `hash_alg`, `signature`	As for a receipt

The issuer fetches statements from the `pool_uri` with a `tip-receipt-request/1` (Section 12.9, item 7) that names the `pool_id` in place of `root`. An auditor or the confidential panel of Section 16.2 is given the pool entries, tags and salts under Section 12.8, item 5, every receipt between `from_log_size` and `log_size`, and a consistency proof. It runs the binding rule of item 2 on every receipt in that range, so a request the site left out of a pool is found. It counts allowed requests whose final receipts show success, together with released results, which are never refunded, and checks the totals. For a sample of receipts it recomputes the tags from the retained results with the issuer's key, so it checks the counting itself and not only that the site agrees with itself. The issuer holds the key and can compute any patient's tag itself when it must answer for that patient. No receipt or statement names a subject. Tags do not match across pools or windows unless a key is reused. Within one window a person's tag is the same at every site of the pool, and anyone who holds both the key and the tags can test identifiers, so the tags and their key are kept as carefully as the identifiers they stand for.

8. **When the counter cannot answer.** A single-site pool is counted by the site itself inside check 12 and depends on nothing else, which is why it works under the interim profile today. A pool counted by a counter service (Section 8.9) is counted only there, never also at each site, and follows its `on_counter_unavailable`. Under `refuse`, a site that cannot reach the service refuses with `counter_unavailable`. Under `safe_state_only`, it allows only declared safe-state actions

(Section 13.5), and only within a local share of the pool. The share is what remained under each limit at the most recent token the site received, divided by the number of sites in the pool's `audience` and rounded down, and in a window for which the site has received no token it is zero. Because each site's latest token may be older than another's, the sites together can pass what truly remained by at most one limit over an outage, and the statements show it. In a hospital, read access for clinical care is often a safe-state action and is exactly what a pool limits, so an outage, or an attacker who blocks the service, never lifts the limit. The site refuses beyond its share with `counter_unavailable`, marks the actions it allows `status_mode: degraded` with that reason code, reports them to the service when it returns, and files each one for review by a person within 24 hours. The service adds them to its counts, and the statements show them in `degraded`. Under the interim profile no break-glass exists, so an agent cannot reach past an exhausted pool, and a clinician's emergency access goes through the site's own procedure for people. Whether a break-glass mandate under Section 9.4 may pass an exhausted pool under the full profile is open question 29, and until it is settled a pool refuses it.

9. **What a pool does not cover.** A pool counts what agents do under mandates of its issuer at sites that acknowledged it. A person working in their own session, and mandates whose root issuer is a staff member's personal TIP-ID, are outside it. An organization that wants those inside uses the site's own access policy, or has its staff's agents work under organization mandates with `sponsor` (Section 18.3). An account linked to two issuers is bound by both issuers' pools, so either can hold back agents acting on it, and the account's owner settles that with the site. A subject known to two systems under identifiers the organization has not linked in `subject_id` counts twice, which overcounts and never lets more through. TIP does not match records of people across systems. The count is the site's own, made from its own declaration of subjects, so it is `site_signed` in the terms of Section 12.10. A site that names the wrong subject for a resource has made a signed misstatement in its endpoint definitions.

8.9 Counter services

A counter service keeps one count for several sites. It serves the `counter` constraint of a mandate (Section 8.5) and every pool whose `audience` names more than one site (Section 8.8). A count it keeps is kept only there. A site keeps no local count of its own share, because a share can never trip the shared limit, apart from the degraded share of Section 8.8, item 8.

1. **Who it is.** A counter service is run by an organization with an active TIP-ID, which may be the issuer itself, such as a hospital counting its own agents across its records, laboratory and imaging systems. It signs with a counter key. Under the interim profile the key is published in an interim key record with the use `counter`, which lists the origin the service answers at, and a reader accepts the record only for an origin the TIP ledger binds to that TIP-ID (Section 23.2, item 9). Later the key is registered through RECEIPT_KEY_REGISTERED with the log kind `counter`. The service publishes endpoint definitions (Section 10.7) whose `pool_uri`,

`reserve_uri` and `settle_uri` say where it takes pool registrations, reservations and settlements, each by POST with the media type of the object sent. It keeps an RFC 9162 log of every token and acknowledgment it signs, signs TIP Seals over it, and under the interim profile registers a seal digest at least once a day (Section 23.2, item 10). It holds counts, mandate and pool identifiers and subject tags. It never receives request bodies, subject identifiers or tag keys.

2. **What it is told.** For a pool, the issuer registers the pool with the service as with a site, without the tag key, and the service returns a `tip-pool-ack/1` signed by its counter key. Its `effective_at` starts window 0 for every site of the pool. For a mandate's `counter`, the first reservation from each site carries, in `mandate`, the mandate that names the service, as it was presented to that site with any disclosures (Section 8.6). The service checks the issuer's signature and that the calling origin is in the mandate's `audience`, and counts against the mandate's `max_uses` and `amount_total` over its whole life, which is window 0. A later reservation for the same `mandate_id` that carries a different mandate body is refused, and both messages are kept as evidence.

3. **Who may call.** Every message from a site is signed by that site's receipt key and carries `created`, an `expires` no more than the request lifetime later, and a nonce. The service accepts it only where the key is bound to the calling origin, through the ledger or an interim key record, and that origin is in the `audience` of the pool or mandate. It answers only on the connection that presented the message and refuses an expired message or a repeated nonce, so one site cannot replay another's reservation. A malformed or unauthorized message gets status 400 or 401 with an `application/problem+json` body and is not logged.

4. **Reserve.** The site sends an object with `typ` `tip-counter-reserve/1`, carrying the site's TIP-ID and origin, the `request_hash`, `mandate` where item 2 requires it, and `entries`, one for each pool and each mandate the service counts for this request. Each entry has `key`, the `pool_id` or `mandate_id`, `window`, and `counted`, an object keyed by limit kind that gives the uses, the value and, for `max_subjects`, the list of subject tags to count. In one linearizable step the service checks every entry against every limit of its key. If all fit, it records a reservation, and if any does not, it records nothing. Either way it returns a token.

FIELD	MEANING
`typ`, `v`	`tip-counter-token/1` and the schema version
`service`	The service's TIP-ID
`outcome`	`reserved` or `refused` for a reservation, `committed` or `released` for a settlement
`reservation_id`	The service's identifier for the reservation
`request_hash`	Copied from the reservation

FIELD	MEANING
`entries`	For each entry, its `key`, `window`, `counted` with tags replaced by their number, and `used`, the totals after the reservation, or before it for a refusal
`refused_key`	For a refusal, the key and limit kind that did not fit
`hold_until`	For a reservation, the time by which it must be settled, within Section 6.6
`log_index`	The token's place in the service's log
`counter_key_id`, `sig_alg`, `hash_alg`, `signature`	The counter key and its signature

A reserved subject tag counts as seen in its window while the reservation is held.

5. **Settle.** The site sends an object with `typ` `tip-counter-settle/1` naming the `reservation_id`, an `outcome` of `commit` or `release`, for a commit the digest of the final receipt and, where the final amount is lower than the amount reserved, `counted` with the lower amounts, and the fields of item 3. A commit may lower what was reserved and never raise it. The service answers with a token whose `outcome` is `committed` or `released`, and a repeated settlement of the same reservation returns the first answer unchanged. A site releases only where the action was refused or failed, as check 13 refunds a local counter. A reservation that is not settled by its `hold_until` is committed, never released, and the service signs and logs a `committed` token for it. A release that arrives later is refused with the reason `reservation_committed`, so a site that crashes or stalls cannot hand back budget it may already have spent. A release carries the digest of the refusal or failed final receipt that justifies it. A release removes a subject tag only where no other reservation or commit in the window holds it. An operation that settles later, such as a payment returned as `accepted_pending`, commits within its declared maximum time and may hold the reservation that long, beyond the default of Section 6.6.

6. **Inside the site's checks.** The site obtains the reservation within check 12, before its local atomic step. If the local step then refuses, the site releases the reservation. For a pool, the reservation token goes into the pool entry's `via`, behind the receipt's `entry_commit`, and for a mandate's `counter` it goes into the receipt. The site keeps the settle token with the receipt under Section 12.8.

7. **One statement for the pool.** For a pool, the service signs a pool statement as Section 8.8, item 7, describes, appended to its own log, covering every site of the pool, with `site` its own TIP-ID, `origin` its origin and log positions in its own log. Each site still signs statements of the requests it made, so an auditor can match the service's totals against the sites' receipts.

8. **What the tokens prove.** A token is the service's signed statement, `stated` in the terms of Section 12.10. Two tokens that both spend the last unit, held by the sites that received them, or a

count that does not match the tokens the sites hold, are signed misstatements that a dispute can use (Section 16.2). The service learns which sites act for an issuer, when, and how many subjects they touch. It never receives who those subjects are. Because tags change with each window, it sees a person's tag recur across sites within one window and never across windows. Where HIPAA applies, a counter service that is not the issuer handles data derived from patient identifiers, and the issuer settles with it the business associate terms that law requires (Section 21.1). An issuer that wants no outside party to learn even that runs its own service. A counter service is a single point of failure for the sites that share it, which is the price of a limit that spans sites, and the pool's `on_counter_unavailable` or, for a mandate, a refusal under Section 8.5 decides what happens when it fails.

theailab.org

9. TIP Approval for high-risk actions

Some actions should never run on standing permission alone. Step-up applies whenever any one of these is true.

1. The verb is listed in the mandate's `step_up`.
2. The action moves money or assets above the mandate's `step_up_amount`.
3. The receiving site has marked the endpoint irreversible, whatever the mandate says.
4. The verb is on the minimum step-up list below, whatever the mandate says.
5. The action would take the cumulative value past the mandate's `step_up_total` within its window.
6. The endpoint is flagged `high_consequence`, the target resource's environment is `production`, and the action is neither `read`, `inspect` nor a declared safe-state action (Section 13.5, item 8).

The TIP Agent Profile publishes a minimum step-up list. It contains every verb whose risk class in the Verb registry is `irreversible`, `financial`, `safety_relevant` or `identity_security`, which includes `delete`, `publish`, `transfer`, provisionally `purchase`, `send_message`, `sign_contract`, `change_credentials`, `change_security_settings`, `change_recovery`, `control_write` and `order_clinical`. Sites apply the minimum list whether or not a mandate repeats it, so a careless or malicious client that leaves it out gains nothing.

A sub-mandate that grants a verb on the minimum list needs its own approval. The parent agent obtains a TIP Approval under the root mandate's `approval_policy` over the hash of the sub-mandate and attaches it as `delegation_approval`, and verifiers refuse the sub-mandate without it.

9.1 The site-signed challenge

When an agent needs a step-up action, it first sends the request in challenge mode, with `intent: challenge`. The site never executes a request in challenge mode. It answers with a **step-up challenge**, signed with the site's receipt key, that names the site's TIP-ID, a `challenge_id`, the HTTP method, the full target URI, the audience, the action, the resource, the `content_digest`, the leaf `mandate_ref`, the approval `meaning`, a `display` list and a short expiry. The site signs the challenge because a hijacked agent must not be able to write its own. A content digest alone is not enough, because many dangerous requests, such as a `DELETE`, have an empty body, and an approval tied only to that digest would match every empty-bodied request.

A person cannot approve what they cannot see. A path such as `/api/v2/orders` or `/fhir/MedicationRequest/88213` tells nobody that the order is for $9,800 or that a dose is changing tenfold, and the client holds only a hash of the body. The `display` list closes that gap. It is an array of typed entries from the Display registry, such as `amount` with value and currency, `payee`, `item` with name, quantity and unit price, `record_count`, `environment`, `subject_ref` and `dose_change` with

from, to and unit. The site derives every entry from the request body it will execute and signs the challenge, display included. The client renders only registered display types in fixed templates and refuses a challenge whose action class needs display types that are missing. Payments need `amount` and `payee`, deletions need `record_count` and `environment`, and clinical orders need `subject_ref` and the clinical change. Because the site signed the display, a display that does not match what the site then executed is the site's own signed misstatement. This is the same dynamic linking that strong customer authentication requires for payments.

The agent forwards the challenge to the approver's TIP app. The client checks the site's signature and TIP-ID, builds the plain-words display from the challenge's typed fields ("Pay $9,800.00 to Acme Supplies Ltd from account ending 4471"), and never shows text the agent or site supplied as free prose. The approver approves with a fresh biometric. The client returns a **TIP Approval** signed by the approver's TIP-ID over the `challenge_id`, the challenge's full hash, the `meaning` and the approver's display name as rendered, and it includes the full WebAuthn assertion, which is the authenticator data, the client data and the signature, with the challenge equal to the SHAKE-256 digest of the approval body. A verifier checks the assertion against a credential public key recorded for that TIP-ID. Where no such key is recorded, the user-verification flag is the client's own statement, and a verifier reports it as `stated`.

The agent resubmits the request with the approval attached. The resubmitted request carries a fresh nonce, so the approval binds the challenge rather than the request hash. The site checks that the approval's challenge was issued by this site, has not expired and has not been used, that the approver satisfies the root mandate's `approval_policy`, and that every field in it matches the new request exactly. The site marks the challenge consumed in the same atomic step that records the nonce and spends the counters (Section 11, check 12), so two copies of one approval sent at the same moment cannot both succeed. The receipt carries `display_hash`.

9.2 Where the approval happens

An agent that can drive the screen must not be able to approve for its human. Browser agents and computer-use agents often run in the same browser profile or desktop session as the person's TIP extension, and such an agent could open the prompt and click Approve, or type a device password it saw earlier. The rules are strict. A TIP Approval must come from an authenticator the requesting agent cannot drive. A client must refuse to show an approval prompt in a browser profile or operating-system session where an agent holding the same mandate is active, and must route the approval to a second registered device, such as the person's phone or a roaming security key. A password or PIN typed into the same session never satisfies step-up for a verb on the minimum list. A site may require `approval_authenticator_class` of `separate_device` or `roaming`, and sites in health care, finance, government and critical infrastructure should.

thevaillab.org

9.3 Approval fatigue

Push-approval systems are attacked by flooding a person with prompts until they tap approve. TIP clients and sites must resist this.

1. Sites rate-limit step-up challenges per agent and per mandate. Clients rate-limit prompts per agent and per site, and batch repeated identical prompts into one.
2. A denial is a signal. After a published number of denials within a window, the client offers to revoke the mandate and, by default, suspends it until the principal acts.
3. Every prompt has a "deny and stop this agent" choice that signs a revocation in the same gesture.

This mirrors the human-present mode in Verifiable Intent and the "one assertion, one signature" rule that already governs every TIP signing. A hijacked agent that tries to wipe a database or send money gets as far as a prompt on its principal's phone, and no further.

9.4 Who approves: policies, dual control and break-glass

Real organizations do not approve with one person. Banks use maker and checker. Power and water operators use a two-person rule for setpoint changes. Hospitals need whichever clinician is on call, not the person who signed the mandate. A mandate therefore carries an `approval_policy`.

FIELD	MEANING
`threshold`	How many approvals are needed, for example 2
`approvers`	TIP-IDs, or roles on the organization's ledger roster such as `approver` or `on_call_pharmacist`
`separation`	The person whose request or agent started the action may not approve it
`timeout`	How long the site waits before the request lapses
`escalation`	An ordered list of fallback approvers if the first do not answer
`end_user_required`	Where an end user's money, data or health is affected, that end user must be among the approvers, through a pairwise pseudonymous identifier

Where an end user's own funds are moved, the end user approves, never only the service acting for them, which keeps the design in line with strong customer authentication rules that require the payer to authenticate.

Emergencies need a door that opens fast and leaves a loud record. A site may list verbs as eligible for **break-glass**, such as emergency read access to a patient chart or an emergency shutdown

command. A break-glass approval needs one approver with a fresh biometric and a reason code from the Reason Code registry, carries the meaning `break_glass`, alerts the organization's security contact at once, and creates a review item that must be closed by a second person within 24 hours. Break-glass never applies to transfers of money, changes to credentials or the registration of content.

HIPAA's emergency access procedure (45 CFR 164.312(a)(2)(ii)) is a required safeguard, and an emergency may come when the approval service or the clinician's phone is unreachable. A hospital may therefore issue a **break-glass mandate** in advance, signed under its roster with a fresh check like any mandate. It names a narrow set of break-glass verbs, such as read access to records for treatment, carries a purpose-of-use code, lasts no longer than a shift, and needs no live approval at the moment of use. Each use is receipted with the meaning `break_glass`, alerts the security contact at once and opens a review item. The break-glass mandate keeps working in degraded mode, because emergency clinical access is declared a safe-state action (Section 13.5).

theailab.org

10. Action requests

10.1 What the agent signs

For each action, the agent signer produces a signature that covers the following.

COVERED ITEM	WHY
HTTP method and full target URI	Stops the request being redirected to another endpoint
`audience`, the receiving site's origin or TIP-ID	Stops a request signed for one site being replayed at another
`content_digest`, SHAKE-256 of the exact body bytes as transmitted, before any decoding	Stops the body being altered after signing; see Section 10.3
`mandate_ref`, the leaf `mandate_id`	Binds the action to one chain of authority
`action` and `resource`	What the agent claims to be doing, checked against the mandate
`created` and `expires`	Freshness; the recommended maximum lifetime is 300 seconds
`nonce`	One-time value; the site refuses any nonce it has seen before
`basis_digest`, optional	Commits the agent's operator to what the agent acted on, without revealing it (Section 10.6)
`build_digest`, optional	The operator's label for the build of the agent that acted (Section 7.7)
`agent_log_head`, optional	A salted digest of the size and root of the TIP Agent Log just before this request's entry, so the receipt fixes the log's position without showing the site how busy the agent is (Section 12.9)
`task_ref`, optional	A random value of at least 16 bytes that the signer creates and gives every request of one task, so a reviewer can find that task's receipts together (Section 15.5). It carries no authority
`intent`	`execute`, or `challenge` to ask only for a step-up challenge (Section 9.1)
`sig_alg`, `hash_alg`	Algorithm labels

On HTTP the signature uses the HTTP Message Signatures framework (RFC 9421), with a `Signature-Agent` header in the style of the Web Bot Auth drafts that resolves to the agent's TIP record. The

thetailab.org

covered components must include `@method`, `@target-uri`, `content-type`, `content-encoding`, `content-digest` computed over the transmitted bytes before any decoding, `signature-agent`, and `idempotency-key`, `if-match`, `tip-basis-digest`, `tip-build-digest`, `tip-agent-log-head` and `tip-task-ref` when present. Sites refuse method-override headers on TIP requests. Each TIP field is an HTTP structured field under RFC 9651. `tip-basis-digest`, `tip-build-digest` and `tip-agent-log-head` are strings of `shake256:` followed by 64 lowercase hex characters, and `tip-task-ref` is a byte sequence. The other signed items travel this way. `created`, `expires` and `nonce` are the RFC 9421 signature parameters of those names. `tip-audience`, `tip-mandate-ref`, `tip-action`, `tip-resource`, `tip-intent` and `tip-content-digest` are structured fields that are always present and always covered, and a site refuses a request in which any of them is missing or uncovered. `tip-content-digest` is SHAKE-256 over the body as transmitted and is the digest every TIP check uses. `content-digest` is covered as well where the client sends one. For an ephemeral key, `Signature-Agent` holds the key thumbprint, and for an agent under an interim agent record it holds the thumbprint and the record's CTID. The request signature uses the context string that the registered HTTP algorithm defines. The `TIP/1/` contexts of Section 6.4 apply to TIP objects, so a request signature can never verify as one. Any header the site uses in deciding is either covered by the signature or ignored. The request `audience` is always one exact origin; a mandate's audience may name a site's TIP-ID, which covers the origins bound to it on the ledger. Other transports, such as MCP and A2A, carry the same signed fields in their own message envelopes (Section 18). The post-quantum registrations these standards need are in Section 10.5.

10.2 Passing the chain by reference

ML-DSA-65 signatures are 3,309 bytes and public keys are 1,952 bytes. In base64 a signature is about 4.4 KB. A request signature fits in an HTTP header, but a chain of three mandates with their signatures is well over 13 KB, which exceeds the 8 KB per-header limit many web servers and proxies apply by default. The design therefore sends the request signature in the header and the mandate chain by reference.

1. On first contact the agent sends the full chain, and the public key of any ephemeral agent key in it, in the request body. A site may instead fetch the chain from a URL the agent gives, but only after the request's signature has verified, only over HTTPS to a public address, and only up to a size cap, so the URL cannot be used to make the site attack its own network or amplify traffic.
2. The site verifies the chain once, caches it by `mandate_id` until it expires, and afterward needs only `mandate_ref`.
3. Agent public keys are resolved from the TIP ledger or from the mandate, then cached by key thumbprint.

Chains, stapled status, approvals, attestation results (Section 7.8) and inspection records (Section 16.5) do not fit in headers. They travel in a preflight call to `/.well-known/tip/agent-session`, which returns a short-lived session handle bound to the agent key, or in a multipart envelope, so the site's

own API body schema stays unchanged. Receipts come back in the response body or are fetched by reference. The preflight is a POST of a canonical JSON object with `typ` `tip-session-request/1`, signed by the agent key like any request. It carries `chain`, the mandates root first, `keys`, the public keys of ephemeral agent keys and of agents under interim records, `records`, the interim records relied on with their CTIDs, and, where used, `attestation` and `inspection`. The response carries `session`, an opaque handle of at most 64 bytes, its `expires`, no later than the earliest mandate expiry and no further ahead than the smallest `max_status_age`, and the version and digest of the site's endpoint definitions (Section 10.7). Later requests carry the handle in the `tip-session` field, a covered component. This keeps steady-state requests to one signature header of about 5 KB, a few short TIP fields and one signature check. Cached chains are refreshed within `max_status_age`, together with the issuer keys, agent records and rosters they depend on, so a cache never outlives a revocation or rotation it should have seen.

10.3 Matching is not authorizing

TIP's content fingerprint, CNA-2.2, normalizes text so that a published post still matches its registered draft after a platform changes spacing, case or punctuation. It keeps only letters, numbers and marks. Pull request 383 showed the consequence for actions. TIP's own `hashContent` returns the same value for `balance=+100` and `balance=-100`.

That behavior is correct for content matching and dangerous for authorization. Every action request therefore commits to SHAKE-256 of the exact bytes. Where an action also produces content, the receipt can carry the CNA-2.2 hash as well, so the action links to the content's CTID, but authorization always uses the exact-bytes digest.

10.4 Request results

The receiving site must bind what it did, not only what it was asked. The receipt (Section 12) includes a `result_digest`, a hash of the response bytes as transmitted, or of a representation of the affected resource after the action that the site declares for the endpoint, so a site cannot later claim a different outcome without contradicting its own signature.

10.5 Standards this builds on, and what remains

Most of the registry work this design needs was completed in 2026, and the design uses the registered names exactly rather than inventing its own.

1. **HTTP signatures.** IANA added `ml-dsa-44`, `ml-dsa-65` and `ml-dsa-87` to the HTTP Message Signature Algorithms registry on October 23, 2026, referencing the C2SP `httpsig-pq` specification. TIP agents sign RFC 9421 requests with the `alg` value `ml-dsa-65`, or `ml-dsa-87` under the national security profile below.

2. **JOSE, COSE and X.509.** RFC 9964 (May 2026) defines ML-DSA for JOSE and COSE, RFC 9881 (October 2025) defines ML-DSA identifiers for X.509, and RFC 10073 (November 2026) defines composite ML-DSA certificates. DPoP tokens, COSE Receipts and A2A Agent Card signatures use those identifiers.
3. **Transparency.** The SCITT architecture is RFC 9943 and COSE Receipts are RFC 9942, both June 2026. TIP receipts map to COSE Receipts, and witnesses act as SCITT transparency services.
4. **Digests.** SHAKE-256 is not yet in the HTTP Digest Algorithms registry of RFC 9530, so the exact-bytes digest travels as a covered TIP field until it is registered.
5. **Web Bot Auth.** The working group draft `draft-ietf-webbotauth-httpsig-protocol` defines the `Signature-Agent` header and a key directory. Where a verifier still accepts only Ed25519, agents add an Ed25519 signature alongside ML-DSA-65 as a transitional measure. NIST IR 8547, final on October 27, 2026, deprecates such classical signatures after 2030 and disallows them after 2035, and Executive Order 14412 sets December 31, 2031 for federal post-quantum signatures, so the Ed25519 part is labeled transitional and never satisfies a TIP check.
6. **National security systems.** NSA's CNSA 2.0 suite requires ML-DSA-87 rather than ML-DSA-65 for national security systems, with new acquisitions compliant from January 1, 2027. TIP therefore defines a CNSA profile in which every key and signature in a chain uses ML-DSA-87, and a verifier configured for that profile refuses any ML-DSA-65 link. The profile is a proposal. TIP-IDs hold ML-DSA-65 keys today, so a chain under it needs TIP-IDs issued with ML-DSA-87 keys, which the RFC must define. TIP hashes with SHAKE-256, where CNSA 2.0 names SHA-384 and SHA-512 for general hashing, and a national security deployment must weigh that before it adopts the profile.
7. **Validated modules.** Federal deployments need ML-DSA and SHAKE from FIPS 140-3 validated modules with CAVP certificates, and FIPS 140-2 certificates moved to the historical list in September 2026. Conforming signers, sites, gateways and witnesses in regulated deployments name their validated module.

The IETF opened agent work in the second half of 2026. IETF 127, in San Francisco from November 14, 2026, holds the first session of the "audit" BoF (Agent Use of Delegation and Interaction Traceability), meetings of the newly chartered agentproto working group, whose charter calls for agents authenticated independently of the users who delegated to them with independent revocation and auditable actions, and sessions of the dawn, webbotauth and wimse groups. The WIMSE group adopted `draft-ietf-wimse-aims` on agent authentication on November 2, 2026, and the OAuth group adopted `draft-ietf-oauth-delegate-sd-jwt` on October 30, 2026. The AI Lab intends to bring the mandate and receipt model to the audit BoF and to align mandates with delegated SD-JWT and Rich Authorization Requests, rather than build a parallel format.

10.6 The basis of an action

A receipt proves what an agent sent and what the site did. In an audit or a dispute the next question is why the agent did it. What instructions did it have, which model and version planned the action, and which tool results and documents did it rely on? An Action Request may therefore carry `basis_digest`, a commitment to the material the agent acted on. Throughout this paper, the agent's operator means the controller of a registered agent. For an ephemeral key it means the controller of the nearest registered agent above that key in the mandate chain. An ephemeral key with no registered agent above it never carries `basis_digest`, because no operator can be named, and a site refuses one that does (Section 11, check 9).

1. **What it commits to.** The agent signer builds a basis manifest, an object with `typ` `tip-basis/1` and a `v`, in TIP canonical JSON. Its `entries` list, in the order the agent received or produced them and each with its `kind`, the principal's instructions and the prompts given to the model, the model identifier and version, each tool call and its result, each document or page the agent relied on, and each memory entry it relied on with its source kind (Section 15.4). An item is the exact bytes the agent received or sent. A document or page whose bytes may not be kept is entered by its URL and retrieval time, or by its CTID. Each entry carries a salted commitment, SHAKE-256 over its own random salt of exactly 32 bytes followed by the item, so an item and its salt can later be erased without breaking the manifest (Section 12.8, item 6). `basis_digest` is SHAKE-256 over a fresh random salt of exactly 32 bytes followed by the canonical manifest. Every digest here has a 256-bit output and is written as `shake256:` followed by 64 lowercase hex characters. Nobody can confirm a guess about what the manifest contains, and two receipts never show that two actions shared a basis. When the field is not used it is absent. On HTTP it travels in the `tip-basis-digest` header, which is then a required covered component (Section 10.1).
2. **Who holds the material.** The agent's operator keeps the manifest, every item it lists and every salt in its own log, under the rule in Section 12.8. None of the basis goes to the receiving site, to TIP nodes or to the ledger.
3. **What the site does.** The site checks only that the field is well formed (Section 11, check 1) and copies it into the receipt unchanged. It cannot check what the digest commits to, because it never sees the material, and the digest's value never affects the decision. A site may require the field for an endpoint class it declares in its published endpoint definitions, and sites that receive clinical or financial recommendations from agents should. It enforces that at check 9 of Section 11 and refuses a request without the field with the reason code `basis_required`.
4. **What it proves.** Covered by the agent's signature, `basis_digest` is the operator's signed statement of what the agent acted on, fixed at the moment of the action. Revealing the manifest, its items and their salts to an auditor or to the confidential panel of Section 16.2 (Section 12.8, item 5) lets them check that statement against the receipt, as Appendix B

outlines. It does not prove that the model used that material, or used it well. A false basis is the operator's own signed misstatement and can be disputed (Section 16.2).

`basis_digest` is part of `tip-receipt/1` from version 1 of the TIP Agent Profile. Every conforming site accepts the field and copies it into its receipt, and every conforming verifier and monitor accepts receipts that carry it.

10.7 Endpoint definitions

Many rules in this paper rest on what a site declares about its own endpoints. The site's mapping of an endpoint to a verb, the environment and data class of a resource, the safe-state actions, the attestation verifiers it accepts and whether a basis digest is required all come from there. A declaration that is unsigned, that is not tied to the site's origin, or that can change without trace would let a site, or someone standing in for it, alter what an endpoint is claimed to be.

1. **One signed document.** A site publishes its endpoint definitions as one object with `typ` `tip-endpoints/1`, at `/.well-known/tip/endpoints`, signed by its receipt key.
2. **What the document carries.** The fields are these.

FIELD	MEANING
`site`, `origin`	The site's TIP-ID and the one exact origin this document serves
`version`, `previous_digest`	An integer that only rises, and the digest of the version before it, absent in version 1
`issued_at`, `expires`, `effective_at`	When the document was signed, an expiry no more than 24 hours later, and when this version takes effect
`profile`	`interim` or `full` (Section 23.2)
`key_record`	Under the interim profile, the CTID of the interim key record for the receipt key that signed the document, with the record itself
`ledger_sources`	The TIP nodes the site reads ledger state from, and who runs each
`max_merge_delay`	The log's maximum merge delay, repeated from the key record, which governs

theailab.org

FIELD	MEANING
`preflight_uri`, `stop_uri`, `receipt_request_uri`, `proof_uri`, `pool_uri`	Where the site takes the session preflight, stop notices, receipt requests, requests for log proofs, and pool registrations and statements (Sections 10.2, 23.2, 12.9, 12.2 and 8.8). The preflight defaults to `/.well-known/tip/agent-session`. A site without `pool_uri` does not support pools. A counter service also gives `reserve_uri` and `settle_uri` (Section 8.9)
`pool_kem_key`	Where the site takes pools that count subjects, its ML-KEM-768 public key for the tag key (Section 8.8, item 7)
`attestation_verifiers`, `approval_authenticator_class`	The verifiers and policies the site accepts (Section 7.8) and the authenticator class it requires for approvals (Section 9.2)
`endpoints`	One entry for each class of endpoint, as item 3 describes

3. **What each entry says.** An entry carries `method`, `path`, a template in which `{name}` stands for one whole path segment, and `verb`, the one registered verb (Section 6.5). It carries `environment` and `data_classes`, each a fixed value or a map from a named path segment to a value. It carries the flags `irreversible`, `reversible` with a short statement of how the action is undone, `safe_state`, `compensating`, `tip_required`, `basis_required` and `high_consequence` (Sections 9, 13.5, 13.6, 7.4 and 10.6), and `result_representation`, which says what `result_digest` covers (Section 10.4). It may carry `formats`, the formats the endpoint accepts (Section 15.6). From Version 1.5 it carries `subjects`, which says how the site lists the data subjects of a request or its result (Section 8.8, item 5).
4. **One path, many operations.** An MCP server, a GraphQL API or a JSON-RPC API takes many operations at one method and path and chooses among them from the request body. For such an endpoint the entry names a `selector`, a field of the request body, and each value of the selector is its own entry with its own verb and flags. The site derives the verb from the body it will execute. A request whose selector value has no entry is refused. An endpoint that cannot be split this way takes the highest risk class of any operation it can carry.
5. **Tied to the origin.** A client or an agent signer accepts the document only where the TIP ledger binds the `origin` it names to the `site` it names, the receipt key is one that TIP-ID has published for that origin, the signature verifies and the document is unexpired. It refuses a version lower than one it has already seen for that origin. The response to the session preflight repeats the current version and its digest.
6. **Fixed in the log.** Before a new version takes effect, the site appends a leaf of kind `endpoints` that holds the document's digest to its receipt log. A verifier can then find the version in force at

a receipt's `decided_at` and check the site's own mapping against it. The site keeps every version under Section 12.8.

7. **What the agent side does with it.** An agent signer caches the document until it expires and may ask again with the version it holds, so an unchanged document is not sent twice. It uses the declared data classes for the `onward` rule (Section 15.5). A site decides by the version in force at its decision, whatever version the agent holds.

The document is the site's own statement. It is `stated` in the terms of Section 12.10, and a site that maps a destructive operation to a mild verb has made a signed misstatement that a dispute can use (Section 16.2).

theailab.org

11. Verification at the receiving site

The receiving site runs these checks in this order before anything changes. The order puts cheap checks first, so a flood of junk requests cannot force the site to run expensive signature checks, and it records the nonce only after the signature is proven, so an attacker cannot burn an honest agent's nonces with forged requests.

An agent signature counts as proven only when the key it verifies against is bound to a valid root, either a registered agent's key in effect or a key named by a mandate whose whole chain of signatures verifies to an active TIP-ID. Anyone can invent a mandate naming a throwaway key, so a signature over such a request proves nothing. A request that fails any of checks 1 to 7 gets a plain, unsigned error and nothing is logged, because signing and logging a receipt costs more than verifying and would turn junk traffic into work. From check 8 onward every failure stops the request with a signed refusal receipt that gives the reason. Signed refusals are rate-limited per root issuer and per controller, and above the limit the site returns unsigned errors.

1. **Shape.** Size limits, required fields, known algorithm labels, canonical encoding, no duplicate keys, no unknown fields.
2. **Freshness.** `created` within the allowed clock skew (recommended 60 seconds), `expires` in the future, lifetime within the maximum.
3. **Audience.** The request names this site's exact origin, as scheme, host and port.
4. **Replay pre-check.** The nonce has not been seen within the validity window.
5. **Agent signature.** The request is signed by the key the mandate names or pins, or by the registered agent's key in effect now. When the request carries `tip-basis-digest`, `tip-build-digest`, `tip-agent-log-head` or `tip-task-ref`, that header is a covered component.
6. **Agent status.** A registered agent's record is active and its controller's TIP-ID is active.
7. **Chain.** Every `mandate_id` is recomputed; each mandate's signature is valid under the key in effect now; the root issuer is a TIP-ID; organization signers are on the roster in effect; every hop obeys the narrowing rules in Section 8.4; depth is within the maximum; no `issued_at` is in the future. A chain whose signatures verify and that carries a field or constraint the site cannot evaluate is refused at check 8, with a signed receipt and the reason code `cannot_evaluate` (Section 8.3).
8. **Revocation and expiry.** No link in the chain is revoked or expired, and no link's `issuer_epoch` is below its issuer's current epoch, using status no older than the smallest `max_status_age` in the chain (Section 13).
9. **Scope and underlying authority.** The site works out the verb and resource from its own endpoint definitions, never from the agent's label alone, and refuses the request if the agent's claimed action differs. A `DELETE` sent with the label `update` is refused. The action, resource, audience and the environment of the target resource must fall inside every mandate in the

chain, and the site-enforced constraints of every link must hold (Section 8.3). The site then authorizes the action against the rights of the root issuer's linked `account` under its own access control (Section 8.7). A request that the account itself could not make is refused, whatever the mandate says. Where the site's published endpoint definition requires a basis digest and the request carries none, the site refuses it with the reason code `basis_required`, and it refuses a basis digest from an ephemeral key with no registered agent above it with `basis_not_allowed` (Section 10.6). It refuses `build_digest` or `agent_log_head` from such a key, and any mandate to such a key that carries `builds`, `inspection`, `inference`, `onward`, `formats`, `reach` or `restrict_after`, with `operator_required`. Where a mandate in the chain carries `custody`, `isolation`, `attestation_max_age`, `builds` or `inspection`, the site checks the acting key against it, `isolation` against the Agent Record or interim agent record (Sections 7.7, 7.8, 7.10 and 16.5). It refuses with `custody_not_allowed`, `isolation_not_allowed`, `attestation_required`, `attestation_stale`, `attestation_invalid`, `build_not_allowed` or `inspection_required`. Where the root mandate carries `sponsor`, the sponsor must hold the role `agent_holder` on the issuer's roster in effect now, and the site otherwise refuses with `sponsor_removed` (Section 18.3). Where a mandate names a pool whose `audience` includes the site and that the site has not acknowledged, that has expired or whose issuer is not the chain's root issuer, it refuses with `pool_unknown`, a site that offers no `pool_uri` refuses any chain that names a pool with `cannot_evaluate`, and where a `max_subjects` pool binds the request and the site cannot list its subjects, with `subjects_undetermined` (Section 8.8).

10. **Step-up.** If step-up applies (Section 9), approvals satisfying the root mandate's `approval_policy`, each for an unconsumed challenge this site issued and matching every field of this request, are attached. A request in challenge mode is never executed; it receives a challenge.
11. **Bytes.** The delivered body hashes to `content_digest`. Where the endpoint declares `formats`, the body, once decoded from any content coding the Format registry lists, then parses as one of them under both the registry's parser and the parser that will carry out the request. The site otherwise refuses it with `format_not_allowed` (Section 15.6).
12. **Atomic commit.** In one atomic, linearizable step, record the nonce, mark every approval's `challenge_id` consumed, and check-and-increment every counter in the chain and every pool that binds the request (Section 8.8). Where a counter service counts a pool or a mandate, the site first obtains its reservation (Section 8.9). Where a pool counts subjects from the result, the site computes the result first and releases nothing until this step succeeds (Section 8.8, item 6). If a challenge is already consumed, or any counter or pool is exhausted, nothing is recorded, any reservation is released and the request is refused, with `counter_exhausted` or `pool_exhausted`. Nonce, challenge and counter state for an origin live in one linearizable store, or are partitioned so that each `mandate_id` is served by exactly one partition. The counts of one pool are never split across partitions or origins, and a pool that cannot be held in one store is counted by a

counter service (Section 8.8, item 2). A site that cannot do this declares a separate origin per region.

13. **Intent, act, receipt.** The site appends a signed `intent` receipt to its log, carries out the action, computes `result_digest`, and appends a signed `final` receipt that names the intent. If the action fails, the final receipt says so and the counters and pools are refunded in the same atomic step, with any counter service reservation released (Section 8.9), except the subjects of a result already released, which are never refunded (Section 8.8, item 6). An operation that settles later, such as a payment, returns `accepted_pending`, and its final receipt follows within a declared maximum time.

Checks 7 and 8 hit caches almost every time. A site that has seen a mandate before verifies one ML-DSA-65 signature per request.

theailab.org

12. Receipts, logs and anchoring

12.1 Receipts

Every decision produces a receipt, including refusals, so attempted misuse leaves evidence. A receipt carries `typ` and `v`, the site's TIP-ID, `log_id`, `receipt_key_id`, `phase` (`intent` or `final`) and, on a final receipt, `intent_ref`, the agent, the verb and resource as the site derived them, `content_digest`, `request_hash` (SHAKE-256 of the RFC 9421 signature base followed by the signature bytes), the leaf and root mandate IDs, the decision and a registered reason code, `decided_at`, `result_digest`, `basis_digest`, `build_digest`, `agent_log_head` and `task_ref` copied from the request when it carried them, `runtime_seen` when a mandate required an attestation result (Section 7.8), `inspection_seen`, the CTID and digest of the inspection record the site relied on (Section 16.5), `roster_seen`, the version of the roster the site relied on when a mandate carried `signers` or `sponsor`, `display_hash` for approved step-up actions, the time token for step-up and financial actions (Section 12.5), counter service tokens for a mandate's `counter` (Sections 8.5 and 8.9), while tokens for pools travel only inside the committed pool entry's `via`, `pools_seen` for every pool that bound the request (Section 8.8, item 7), `profile` (`interim` or `full`, Section 23.2), `status_mode` (`normal` or `degraded`), the receipt's `log_index`, and two facts about what the site knew when it decided. `status_seen` names the version of the revocation status it relied on, and `ledger_seen` names the most recent TIP ledger transaction it had observed. The site signs it with a receipt key registered under its TIP-ID.

The site returns the signed receipt in the response to the agent, synchronously. A site that claims TIP agent conformance must do this. The agent keeps its own copy, so evidence of the action exists in two places held by two different parties. On HTTP the response carries `tip-receipt`, a structured field with the digest of the final receipt and a URI under the site's origin where the agent fetches the intent and final receipts within the request lifetime. A site may instead return a multipart response whose second part is `application/tip-receipt+json`. A failure at checks 1 to 7 returns status 400 or 401 with a body of `application/problem+json` whose `type` names the check. It is unsigned and is not logged.

12.2 The receipt log

Each site appends receipts to a Merkle log. The log uses the tree shape and proofs of RFC 9162, Certificate Transparency version 2, with inclusion proofs bound to the leaf index and tree size and consistency proofs between any two sizes, using SHAKE-256 with the RFC 9162 leaf and node prefixes registered as a TIP hash profile. TIP's current `shared/merkle.js`, which pull request 383 uses, builds a different tree and has no consistency proofs, so it must be replaced before witnesses can work. The receipt format is aligned with SCITT (RFC 9943) so transparency tooling can read it. At intervals the site signs a **TIP Seal** containing the log size and root hash. A seal is an object with `typ` `tip-seal/1` that carries `log_id`, `size`, `root`, `sealed_at`, `receipt_key_id` and the digest of the

previous seal. A leaf is the SHAKE-256 digest of the canonical bytes of a signed object, which is a receipt, a stop acknowledgment, a pool acknowledgment, a pool statement or an endpoint definition digest, under the RFC 9162 leaf prefix. A site serves inclusion and consistency proofs at the `proof_uri` in its endpoint definitions (Section 10.7). Appending a leaf, signing a seal and producing an inclusion or consistency proof take work that grows with the logarithm of the log's size, never with its size. An implementation keeps the hashes of the tree's complete subtrees, at most one for each level of the tree's right edge, together with the interior hashes it needs for proofs, and never rebuilds the tree to add one leaf. A log that rebuilds on every append slows as it grows, and because a site appends a receipt before it answers (Section 11, check 13) and an agent signer appends before it releases (Section 12.9, item 4), a slow log slows every decision and in the end stops the signer.

12.3 Witnesses stop split views

A site that controls its own log could show one version to one auditor and a different version to another. Independent **witnesses** prevent that. Each log has a published set of at least three witnesses, assigned under rules the AI Trust Council sets rather than chosen by the site, and no witness may share ownership or control with the site, or host the site's log or evidence store (Section 20, item 10). A witness must hold an active organization TIP-ID, so nobody can create witnesses at will. A change to a log's witness set is a WITNESS_SET_UPDATED transaction cosigned by a majority of both the old and the new sets, which keeps the guarantee below intact across the change. A witness cosigns a TIP Seal only after checking that it extends the last TIP Seal it cosigned for that log (a consistency proof). A TIP Seal counts only with cosignatures from a majority of that log's witness set. Because any two majorities share at least one witness, and an honest witness never cosigns two TIP Seals that do not extend each other, two conflicting histories cannot both be accepted unless a witness signs both, and that pair of signatures is itself public proof of misconduct.

Witnesses do not sign every site TIP Seal one by one. Each witness verifies consistency for every TIP Seal it receives during an interval, puts them in a Merkle tree keyed by log identifier, and signs the tree's root once. A site's witnessed TIP Seal is its TIP Seal plus inclusion proofs under a majority of those signed roots. This keeps a witness's signing work to one signature per interval, however many sites it serves.

Someone has to look. A **monitor** is a conformance class that fetches every anchored seal, checks consistency across all of them, and publishes any fork it finds as a signed proof. The AI Lab and at least two independent parties run monitors, and anyone may.

If a log cannot obtain a witnessed seal within its maximum merge delay, because too many witnesses are unreachable, the site refuses every action except those it has declared as safe-state actions (Section 13.5) until it can.

theailab.org

12.4 Anchoring on the TIP ledger without overloading it

Witnessed roots are anchored on the TIP ledger, which gives every log a single public history and an independent clock. Anchoring every site's TIP Seal as its own transaction would not scale. With 100,000 sites sealing every five minutes, that alone would be 28.8 million transactions a day, about 333 per second, more than three times the 100 per second at which the TIP specification plans to move to full decentralization.

The design anchors in two tiers. Aggregators, which are typically the witnesses themselves, anchor one signed root per interval with a new LOG_CHECKPOINT_ANCHORED transaction. A thousand aggregators anchoring every five minutes produce 288,000 transactions a day, about 3.3 per second, however many sites and actions sit underneath. Each aggregator tree is keyed by log identifier and holds at most one TIP Seal per log per interval, so for any log the anchored history is a single sequence, and two different TIP Seals of the same size for the same log, anchored anywhere, are public proof that the log forked.

A site's inclusion proof runs from receipt to site TIP Seal to witness root to ledger transaction. Every conforming log has a **maximum merge delay**. A receipt must appear in an anchored TIP Seal within that time. The default is 15 minutes and the upper limit 24 hours, and each log states its value in its registration. Busy sites TIP Seal by count and by time, whichever comes first; low-traffic sites may seal less often, but never beyond the maximum merge delay.

12.5 Time

A receipt's `decided_at` comes from the site's clock, which pull request 383 lists as a limit. Ledger time means the commit time attested by the committing quorum's signatures, the coordinator in TIP's Phase 1 and the committee in Phase 2, never the timestamp a transaction's signer chose. The ledger bounds `decided_at` from both sides, and `ledger_seen` must be fresh at `decided_at`, so a site cannot weaken the lower bound by naming an old transaction. Fresh means that the ledger committed no later transaction more than 60 seconds before `decided_at`. A quiet ledger therefore never makes an honest receipt fail, and a verifier checks the rule from the ledger's own history. For step-up and financial actions the site also obtains, before committing, a time token signed by a threshold of TIP time signers over the `request_hash`, and puts it in the receipt. It cannot be earlier than the ledger time of the `ledger_seen` transaction the receipt names, because the site claims to have already seen that transaction. It cannot be later than the ledger time of the anchor that includes the receipt, and that anchor must arrive within the maximum merge delay. A verifier checks both bounds and the request's own `created`. The width of the window is set by how recently the site last read the ledger and by the merge delay, so high-value sites read the ledger often and keep both short.

12.6 Withheld receipts

A site could act and never log the receipt. Three things narrow that gap. The receipt is returned synchronously, so an agent that received a response without a receipt knows at once and records the fact in its own log. After the maximum merge delay, the agent, or anyone holding a receipt, asks the site for that receipt's inclusion proof against an anchored TIP Seal; a site that cannot produce it has been caught withholding or forking, and the signed receipt is the evidence. Agent operators keep a TIP Agent Log (Section 12.9), witnessed and anchored the same way, so the agent side holds a signed request and a timestamped record of any missing receipt. Withholding cannot be prevented by cryptography alone, but under this design it leaves evidence.

12.7 Receipt and witness keys

A stolen receipt key lets an attacker forge receipts for new decisions and forge the challenges that drive approvals, so these keys need a full lifecycle. Receipt and witness keys are held in FIPS 140-3 Level 3 or equivalent hardware, declared as `key_custody` when registered. RECEIPT_KEY_ROTATED replaces a key on schedule. RECEIPT_KEY_REVOKED carries `compromised_since`, and receipts or challenges signed after that time are invalid unless they appear in a TIP Seal anchored before the revocation transaction itself. A key owner therefore cannot disown receipts that were already anchored by naming an early time. Witness keys follow the same rules through WITNESS_KEY_REVOKED.

12.8 Keeping the evidence

A digest proves that a record still matches what was signed. Once the record is gone, the digest proves only that something was signed. Every party therefore keeps the originals behind the digests it signs or relies on, and TIP holds none of them.

1. **Who keeps what.** The receiving site keeps each signed request in full as it received it (the covered components, the signature input and the signature) and its body, the mandate chain exactly as presented with any disclosures, delegation approvals, approvals with the challenges and displays it signed for them, stapled status statements, counter and time tokens, the results it produced with the representation of the affected resource that `result_digest` covers (Section 10.4), and its signed receipts. The agent's operator keeps its signed requests in full, the chains it presented, its copies of receipts, and the basis manifest, items, salts and any erasure records behind each `basis_digest` (Section 10.6). The operator also keeps each build statement with its salt (Section 7.7), the originals behind its TIP Agent Log entries (Section 12.9), its transfer statements (Section 7.9) and its transcripts, for their own period (Section 7.10, item 7). The agent signer keeps the bytes it refused under Section 15.6 in an evidence store apart from memory, and the operator keeps no other copy of them. The site also keeps each attestation result and inspection record it relied on (Sections 7.8 and 16.5). A site that counts pools keeps each pool with its acknowledgment, the tag key, each pool entry with its salt, the

tags and salts behind every commitment, and its pool statements, and a counter service keeps its tokens with the messages they answer (Sections 8.8 and 8.9). Principals and approvers should keep copies of the mandates and approvals they signed, so they never depend on the other side's copy in a dispute, and their TIP client keeps them where it can. These copies are advised, not required, and carry no retention period.

2. **Where.** In systems the party controls, its own or a provider's it contracts with, in the region its law requires. TIP does not move them, and no witness, node operator or aggregator receives them in that role. A party that uses a hosted evidence store stays responsible for keeping its evidence (Section 20, item 10). Inside an enclave, originals stay inside it except for disclosure under item 5 (Sections 18.1 and 21.3).
3. **For how long.** At least the retention in Section 6.6, which binds sites and operators, and longer where law or contract requires. A party that deletes an original early, other than for erasure under item 6, can no longer prove that action.
4. **How.** Originals are stored as the exact bytes their digests cover, as transmitted before any decoding, so they can be matched later. They are encrypted at rest, with access to them logged. Personal values and their salts are stored apart (Section 20, item 6).
5. **Disclosure.** Originals go only to the parties to a dispute and its adjudicators under Section 16.2, to auditors and regulators entitled to them, or under legal process. The principal's own TIP client may also show the principal the bytes a signer refused, where Section 15.6, item 8, allows, and the parts of a transcript that Section 7.10, item 7, allows. Evidence that contains regulated data, such as health records or customer financial data, and every basis manifest with its items, goes only to the confidential panel of Section 16.2, never to community jurors. The recipient checks what it received against the digest in the signed receipt.
6. **Erasure.** Erasure never overrides a legal duty to retain or a legal hold. Erasure includes the sanitization of classified or controlled information that reached a system not accredited for it, recorded in the same way. Where it applies, deleting a personal value and its salt meets the duty, and the receipt stays valid but no longer proves that value. A value inside a request body is erased by deleting the whole body, together with the random value that Section 20, item 6, requires in bodies that carry short personal values, so the receipt's `content_digest` can no longer confirm a guess. Each erasure is recorded with the time it happened, signed by a system key the party names, and anchored in the party's own witnessed log (Sections 12.2 and 12.9). A party without such a log fixes the record's time another way it can later show, such as a TIP content registration of the record's hash. In a dispute, an original missing before its retention period ended counts against the party that held it, unless its erasure record was fixed in time before the dispute was filed.
7. **Values nobody may keep.** Some rules forbid keeping certain values at all, such as card security codes under PCI DSS. Endpoints that handle such values must receive them by token or by reference, so the signed body never contains them. An endpoint that receives such a value in a

signed body does not conform, because the receipt's `content_digest` could confirm a guess even after the body is deleted.

12.9 The TIP Agent Log

Receipts exist only where the receiving site issues them. Most sites an agent touches today have never heard of TIP, and an action there leaves no evidence that anyone but the agent's operator can produce. Earlier versions said in Section 12.6 that operators can run their own logs. This section defines that log.

The agent's operator (Section 10.6) keeps one **TIP Agent Log** for each registered agent. Section 7.4 already requires that the agent's only way out is through the agent signer or its policy proxy, so that is where the log is written.

1. **What goes in.** Each entry has `typ` `tip-agentlog/1`, a `kind` from the Agent Log Entry registry, the time, the most recent TIP ledger transaction the signer had seen, a random salt of exactly 32 bytes, and digests, never content. A `request` entry holds the `request_hash`, audience, leaf mandate, `build_digest` and `task_ref` of a signed Action Request. A `receipt` entry holds the hash of the receipt that came back, and a `no_receipt` entry records that none did. An `unreceipted_action` entry records a request to a site that does not speak TIP (item 3). An `inference` entry records material sent to a model (Section 18.2). A `memory_write` entry records a write to memory (Section 15.4). A `flow` entry records a decision under the `onward` constraint (Section 15.5). A `state` entry records a change of signer state, or of a mandate's restriction flag together with the mandate, and a `late` entry records an action released while the log was down, with both times (Section 7.8). `transfer`, `unlock`, `inspection_access`, `coverage` and `erasure` entries record those events (Sections 7.8, 7.9, 12.8 and 16.5). A `delegation` entry records a sub-mandate the signer issued (Section 7.4), and a `stop` entry records a stop notice it received (Section 23.2). A `format_anomaly` entry records a refusal under the `formats` constraint, unreadable material and a language mismatch (Section 15.6). It names the `cause`, which is `format`, `unreadable` or `language`, and the destination, as `target_origin` or `inference_place`. `mandate_id` names the leaf mandate. Where the request names none, the signer writes one entry for each mandate the refusal counts against, or one entry with `mandate_id` `none` where it counts against none. `allowed` is an object with `from`, which is `mandate` or `endpoint`, and `entry`, the format entry that applied. `detected` is the format the signer found or `unknown`, and `encoding` names an encoding found inside it. `part` is `body`, `uri`, `query` or `header`, and the optional `pointer` locates the material within the part, as a JSON pointer (RFC 6901) inside JSON, as the header name or query parameter name, as `path` for the target URI's path, and otherwise as a byte range `bytes=first-last` within the part. `refused` says whether the request was refused, `refusals_in_window` gives the counted refusals against the mandate in the current window, and `material_digest` is a digest of the material, never the material. The entry's salt is the first input to every digest in the entry. `code_load`,

`containment_denied`, `transcript` and `containment_test` entries record the events of Section 7.10, items 5, 6, 7 and 10, and the actions of a scope reviewer and a halt are `state` entries. A signer writes a `receipt` entry only after the receipt's signature verifies against the site's receipt key, and a `no_receipt` entry otherwise.

2. **Each receipt fixes the log's position.** A conforming agent runtime sends `agent_log_head` in every request, covered by the agent's signature (Section 10.1). It is SHAKE-256 over a salt of exactly 32 random bytes, fresh for each request, followed by the canonical JSON object that holds the `root` and `size` of the log as last durably appended when the signer signed. The size in a head is no smaller than the index of the `request` entry of any earlier request signed by that key, and the request's own entry sits at that size or later. A verifier that opens two heads from one key checks that order against the receipts' `decided_at`. A head that breaks it, such as the head of an empty log sent with every request, is a finding under Section 16.2. The salt keeps a site from reading the log's size, which would show how busy the agent is everywhere else. Requests released at the same moment may carry the same head. The receiving site copies the field into the receipt. A principal's client, a monitor or an adjudicator who is given the receipt asks the operator for the salt, the size and the root, checks them against the digest, and asks for an RFC 9162 consistency proof from that size and root to a later sealed or anchored head. An operator that cannot give them has removed or rewritten earlier entries.
3. **Sites that do not speak TIP.** Version 1 of the profile records what an agent does at such a site and authorizes nothing there. A mandate's `audience` names only sites that speak TIP. Its `reach` constraint limits where the agent may connect (Section 7.10), and the site's own access control governs what the agent may do there, through credentials the agent holds or, inside a boundary, only a credential broker inserts. The agent's traffic to the site still passes through the signer's policy proxy (Section 7.4). The proxy writes an `unreceipted_action` entry with the site's origin, the method, the time and digests of the request and the response. It applies the `onward` rule to that traffic (Section 15.5), and because a plain request names no mandate, it applies the strictest rule among all principals whose memory partition or mandate is loaded in the task, which inside a boundary of Section 7.10 are one principal's and treats a mandate with no `onward` as naming no class. The proxy never decides that a site does not speak TIP from a failed fetch. It refuses any unsigned request to an origin for which it holds, or has ever held, endpoint definitions, and it refuses until a fetch of the definitions succeeds or returns a clear not-found over a verified connection. The signer holds no credential for the site and applies no mandate's authority there, only the limits of `reach`, `onward` and `formats`. An earlier draft of this version had the signer hold those credentials and apply a mandate on the agent's side. That put every principal's ordinary credentials in one place and let the agent side stand in for a site, so it is left out of version 1. Open questions 22 and 23 ask whether a later version should allow it.
4. **No record, no release.** A conforming signer releases a request only after its entry is durably appended. If the log cannot be written, or cannot obtain a witnessed seal within its maximum merge delay, the signer moves to `restricted` (Section 7.8). It then releases only actions the

target site has declared safe-state actions (Section 13.5) and writes a `late` entry for each as soon as the log returns.

5. **Sealed like a site log.** The log is an RFC 9162 tree like the receipt log of Section 12.2, so a change to a sealed entry breaks its seal. The operator signs TIP Seals with a log key registered under its TIP-ID in the way a receipt key is registered. RECEIPT_KEY_REGISTERED therefore gains a log kind, `site` or `agent`, and the agent the log serves, as part of that proposal. Until the transaction exists an operator publishes its log key in an interim key record (Section 23.2, item 9). The log key is held in the same custody class as the agent key, and a verifier reports that class. The rules of Section 12.7 on rotation, revocation and `compromised_since` apply to it. The hardware requirement of that section does not. A conflicting seal signed after the `compromised_since` time is not evidence against the operator. Witnesses assigned under Section 12.3 cosign the seals, and aggregators anchor them under Section 12.4. The log has a maximum merge delay (Section 6.6) and may pad its seals as the volume privacy item of Section 20 describes.
6. **Who may read it.** A principal may read the entries made under that principal's mandates, and nobody else's. An inspector reads entries within the scope of an inspection (Section 16.5). Originals behind the digests follow Section 12.8.
7. **Coverage.** At each seal the operator writes a `coverage` entry for each root mandate with activity, giving the number of actions taken under it and the number of those that have a receipt from the site. This is the operator's own count. A principal checks it from the other side. A receiving site gives the root issuer of a mandate, on a request signed by that issuer's TIP-ID, the receipts it issued under that mandate. The request is an object with `typ tip-receipt-request/1`, signed under Section 8.2 and sent to the `receipt_request_uri` in the site's endpoint definitions (Section 10.7). It carries `issuer`, `audience`, which is the one origin it is sent to, `root`, the time range `from` and `to`, `created`, an `expires` no more than five minutes later and a nonce. A site refuses a request that names another origin, has expired or repeats a nonce, and it answers only on the connection that presented the request, so one site cannot replay it at another. The answer carries the receipts, the site's latest seal and an RFC 9162 inclusion proof for each receipt against that seal.
8. **What it proves.** An anchored entry proves that the operator recorded this action no later than the anchor and has not changed the record since. Where a receipt from the site exists, the two records check each other. Where none exists, the entry is one party's statement about itself, and Section 12.10 labels it `one_party`. An entry that was never written leaves no trace in the log, so the log cannot show that an unlogged action did not happen. Between a request and the next seal, entries for sites that do not speak TIP rest on the operator alone. A receipt, a counter token or a site's own record that has no matching entry is evidence that the log omits actions, and that is a finding under Section 16.2.

The signer and its policy proxy see request content in the clear and are a valuable target. The limits are stated in T67 and T81. An ephemeral key with no registered agent above it has no operator, so it has no TIP Agent Log, and evidence of its actions exists only at the sites it used.

12.10 TIP Evidence States: who stands behind a record, and how firmly it is fixed

The records in this design do not all prove the same amount. Earlier versions described each kind in its own section. This section names two facts about any record, so that a verifier can report them and a reader does not mistake a claim for a proof. The first is who stands behind the record. The second is how firmly it is fixed in time. They are separate. Fixing a record in time does not make it true.

WHO STANDS BEHIND IT	WHAT EXISTS	WHAT IT SHOWS	WHAT YOU STILL TAKE ON TRUST
`stated`	A party's own description of something, such as an agent manifest, a mandate's `purpose`, a build statement, an Agent Card, a coverage count or an inspection record	Who made the claim, where it is signed or committed to under a signature	That the claim is true
`agent_signed`	An Action Request signed by the agent key	That this agent key asked for exactly this, under this mandate	Everything about what happened next
`site_signed`	A TIP Receipt signed by the receiving site	The site's signed statement of what it decided about that exact request	The site, for what it then did, for `result_digest`, for its mapping of the request to a verb and an account, and for its clock

HOW FIRMLY IT IS FIXED	WHAT EXISTS	WHAT IT SHOWS	WHAT YOU STILL TAKE ON TRUST
`held`	Only the parties hold the record	Nothing beyond the signatures on it	That neither party issued a different record for the same event, or dropped this one
`sealed`	The record is inside a signed TIP Seal of its log	That the log held the record when the seal was made, and a later change breaks the seal	The log's owner, not to show different seals to different people
`witnessed`	A majority of the log's witnesses cosigned that seal	That the log has one history, unless the owner colludes with the witnesses that two majorities share, which is one witness in a set of three	That no such collusion happened

thealab.org

HOW FIRMLY IT IS FIXED	WHAT EXISTS	WHAT IT SHOWS	WHAT YOU STILL TAKE ON TRUST
`anchored`	The witnessed seal is included in a TIP ledger transaction	That the record existed no later than the ledger time of that transaction, in one public history	The TIP ledger's own guarantees, including its single coordinator until Phase 2 (Section 13.5), and the signature algorithms

A TIP Receipt is `site_signed` and moves from `held` to `anchored`. An entry in a TIP Agent Log is first signed when the operator seals its log, so it is `stated` by the operator and moves from `sealed` to `anchored`. However firmly it is anchored, it stays one party's account. An anchored record can be checked by anyone who holds the record and its proofs. The ledger holds only roots.

The first scale describes records of actions. A mandate, a TIP Approval, a transfer statement, an attestation result and an inspection record are each one named party's signed statement. A verifier reports them as `stated`, names the signer, and judges what they authorize or describe by the rules of their own sections.

Qualifiers travel with a record and never change its states.

QUALIFIER	MEANING	SECTION
`one_party`	The record comes from a TIP Agent Log and no verified receipt from a site matches it	12.9
`related_party`	The site that signed the receipt is the agent's controller or the mandate's root issuer, or shares a beneficial owner with one of them, so the two signatures do not come from independent parties	16.2
`approved`	A TIP Approval covers the action, and the receipt carries the hash of the display the site signed for it	9
`time_token`	A threshold of TIP time signers bounded the decision time	12.5
`degraded`, `break_glass`, `compensating`, `pending`	The site decided under degraded status, under break-glass, as a compensating step, or the final receipt has not yet followed	13.5, 9.4, 13.6, 11
`enclave_witnessed`	The seal's cosignatures come from witnesses inside the facility, which do not count toward the majority of Section 12.3. Provisional until open question 13 is settled	18.1

thetiap.org

QUALIFIER	MEANING	SECTION
`runtime_seen`	The site says it relied on an attestation result for the acting key. A verifier who is given the retained result can check it	7.8
`agent_custody`	The custody class of the acting agent key, marked `declared` or `attested`	7.3, 7.8
`agent_isolation`	The isolation class of the acting agent, marked `declared` or `measured`	7.3, 7.10

The verifier of Appendix B returns `valid` only for a receipt that is `site_signed` and `anchored`. For any record it also reports both states and the qualifiers, and it lists the constraints in the chain that it could not check, so a result of `indeterminate` still says how far the evidence reached.

Two things are not evidence of an action. An inspection (Section 16.5) is a named party's opinion about an agent, and a verifier reports it as `stated`. A refusal by the agent signer or by a site prevents an action. Evidence only records one.

theailab.org

13. Revocation and TIP Stop All

13.1 What can be revoked

TRANSACTION	EFFECT
MANDATE_REVOKED	Revokes one mandate by `mandate_id`, and every sub-mandate under it
MANDATE_EPOCH_ADVANCED	Raises the issuer's integer revocation epoch; every mandate carrying a lower `issuer_epoch` stops working, in one transaction; this is TIP Stop All
AGENT_REVOKED	Revokes a registered agent; every mandate naming it stops working
AGENT_SUSPENDED	Suspends a registered agent for new actions, by its controller or by an emergency panel (Section 13.7)
ROSTER_UPDATED	Removing a member ends that member's signing and approving roles and every mandate that names the member as `sponsor` (Sections 8.2 and 18.3)
MANDATE_EPOCH_RESTORED	Lifts a suspension-type stop, restoring the previous epoch only (Section 13.6)
Existing TIP revocations	A revoked or device-revoked TIP-ID can no longer authorize; mandates it issued stop working

TIP Stop All uses an integer epoch, not a date, because a date inside a mandate is chosen by whoever signs it, and a thief with a stolen key could sign with a future date. An epoch can only be raised by a ledger transaction, so it cannot be predicted or skipped.

Revocation is signed by the issuer of the revoked mandate, by any issuer higher in the chain, or, for agents, by the controller. An organization can also let a designated security contact on its roster stop agents without the usual signing quorum, because stopping an agent should be easier than starting one. A stop signed by a single security contact suspends rather than revokes (Section 13.6), so one compromised contact cannot permanently destroy every mandate an organization holds.

13.2 Checking status at scale

Asking a ledger node about every link of every chain on every request would cost the network money on every action (Section 22), make every site depend on a live connection, and tell node operators which principals use which sites. The design uses four layers.

1. **Short lifetimes.** Mandates for routine work last hours, not months. Expiry does most of the revoking.
2. **Status entries.** Every mandate that can be revoked on the ledger carries a `status_ref` and a `status_index`, a position in a status list in the style of the W3C Bitstring Status List, in which each mandate holds one bit. Lists are shared pools of at least 131,072 entries holding many issuers' mandates, positions are assigned at random, and `status_ref` names the pool, never the issuer, so a status reference does not reveal who signed the mandate. Issuers obtain positions in blocks of at least 4,096 per ledger transaction, so status allocation does not add a ledger write per task. Sites fetch whole pools or rely on stapling, never per-entry queries. One list covering 131,072 mandates is 16 KB before compression. An ephemeral agent key cannot hold a list, so sub-mandates it issues rely on short expiry and on revocation of their parent.
3. **Stapled status.** The agent attaches a recent signed statement of the current status of every link in its chain, in the way TLS servers staple certificate status. Not any node may vouch. Status statements are valid only when signed by a threshold of status signers, by default two of three, the three run by independent operators in different jurisdictions, registered by STATUS_SIGNER_REGISTERED transactions that the AI Trust Council approves. Each status list version carries a strictly increasing version number, the hash of the previous version and the ledger transaction that produced it, and status list roots are anchored on the ledger at a fixed interval. The site checks the threshold signature and the statement's age instead of looking anything up. This removes most status lookups and keeps node operators from learning which sites a principal's agents visit.
4. **A global change feed.** Nodes publish one signed feed of all revocations, epoch changes and roster changes. Sites that prefer not to rely on stapling subscribe to the feed and keep their own current view, with no per-principal polling.

Each mandate states `max_status_age`. A site may rely on status, stapled or cached, no older than the smallest `max_status_age` in the chain. `status_seen` in a receipt names the status version and carries its threshold signature, and a verifier checks that no anchored status version issued between the one the site used and the decision, older than the allowed age, showed the link revoked. A mandate for routine reads might allow ten minutes; a mandate that allows payments might allow thirty seconds.

13.3 Fail closed

If a site cannot obtain status fresh enough for the mandate, it refuses ordinary actions. There is no "allow if the status server is down" mode for actions that change something, and step-up actions always need status checked at the time of the action. A site may serve reads under stale status only if the mandate's `max_status_age` permits it. Section 13.5 sets out the one exception, safe-state actions, which exists because refusing everything can itself cause harm.

theailab.org

13.4 History stays true

An action decided before a revocation took effect stays valid afterward, so revoking an agent never rewrites history. The test is what the site was entitled to know. An action is valid if, at the site's decision, the status version named in the receipt's `status_seen` showed no revocation, that status was within the chain's `max_status_age` of `decided_at`, and the keys were the keys in effect at `decided_at`. A site that acted on status older than it was allowed to use, or ignored a revocation its own `status_seen` already contained, carries the responsibility for that action.

13.5 Safe state and degraded operation

A design that refuses everything when it cannot reach the network is safe for a bank's marketing site and dangerous for an intensive care unit or a water treatment plant. Lost availability in clinical and industrial settings is a safety failure, not an inconvenience. This section is therefore normative for every site in health care, energy, water, transport and government emergency services, and recommended everywhere.

1. **Safe-state actions.** A site declares, in its published endpoint definitions, the actions that move the controlled system to a documented safe condition, such as stopping a pump in a defined way, holding a process at its last safe setpoint, or read-only access to a patient record for clinical care. These are the only actions it may allow under degraded status.
2. **Mandate choice.** A mandate's `on_status_unavailable` is either `refuse` or `safe_state_only`. Under `safe_state_only`, when status older than `max_status_age` is all the site can get, the site refuses everything except declared safe-state actions. A stale statement stapled by the agent is never a reason for degraded operation. A site enters it only when its own status sources, the change feed and the status signers, cannot be reached, because an agent that chose what to staple could otherwise choose the mode.
3. **Loud records.** Every action taken under degraded status gets a receipt with `status_mode: degraded` and a reason code, and the site files each one for review by a person within 24 hours.
4. **No single status source.** Status must be obtainable from at least three independently operated status signers in different jurisdictions (Section 13.2). No single operator, The AI Lab included, may be required for a status answer.
5. **The ledger today.** Until TIP reaches its Phase 2 committee, ledger commits pass through one coordinator. No deployment in a safety-critical sector may depend on ledger liveness for safe-state actions, and this paper describes the current ledger as coordinator-committed.
6. **Stops hand control back.** In industrial settings a stop returns the process to traditional automation or manual operator control. It never halts the plant. This follows the joint guidance led by CISA with partner agencies in Australia, Canada, Germany, the Netherlands, New Zealand and the United Kingdom, Principles for the Secure Integration of Artificial Intelligence in Operational Technology (December 3, 2025), which calls for failsafes that revert to manual control, operation without continuous internet or cloud access, and AI identities logged

separately from other identities. IEC 62443-3-3 says access controls shall not prevent essential functions, including safety functions, from operating, and OMB M-25-21 recognizes that human approval is not practicable for an automated safety mechanism that must act at once.

7. **Never in the loop.** TIP is never placed inside a safety instrumented function or a real-time control loop. It governs requests that reach a control system through an enforcement gateway (Section 18.1) and stays out of the path of protective functions.
8. **A person at every change to a high-consequence system.** A site flags as `high_consequence`, in its endpoint definitions (Section 10.7), every endpoint that acts on a system whose failure can endanger life, health or essential services, such as clinical systems, industrial control, energy, water and transport systems, and defense systems. A site in a sector this section names flags every such endpoint. An endpoint that only schedules, informs or takes requests, such as an appointment booking service, need not be flagged. On a resource whose environment is `production`, a flagged endpoint accepts `read`, `inspect` and declared safe-state actions on standing permission. Every other action there needs a TIP Approval (Section 9, rule 6) under an `approval_policy` with a `threshold` of at least two and `separation`, given on devices the agent cannot drive (Section 9.2), and the site refuses an approval under any weaker policy with `step_up_required`. A break-glass verb at a flagged endpoint is allowed only where the site declares it a safe-state action. Agents that develop or test changes to such systems work on resources whose environment is `replica` or `testbed`. Under the interim profile no TIP Approval exists, so the site refuses every other action on such a resource (Section 23.2, item 5), and the agent prepares the change for a person to make in their own session. Section 7.10, item 3, keeps an agent from reaching such a system around its gateway.

13.6 Scoped stops, suspension and restoration

TIP Stop All is powerful, which makes it a target. Three rules keep it from becoming a weapon.

1. **Scope.** MANDATE_EPOCH_ADVANCED carries `scope`, which is `all`, one agent, one verb, or one environment label. Mandates carry `epoch_scope`, so an organization can stop every agent touching production, or one compromised agent, without stopping its whole fleet.
2. **Suspension first.** A stop signed by a single security contact suspends the affected mandates. The suspension lapses after 72 hours unless the organization's normal signing quorum confirms it, and the quorum can lift it at once with MANDATE_EPOCH_RESTORED, which can only restore the previous epoch. A stop signed by the full quorum revokes.
3. **Finishing safely.** An operation already under way when a stop arrives, such as a database migration, a valve sequence or a transfer debited but not yet credited, may still run the steps the site declares as compensating or safe-state steps for that operation, for up to the compensation window (300 seconds by default). Every such step is receipted as `compensating`.

Stop All does not answer a stolen principal key. A thief who holds the key can read the new epoch and sign fresh mandates at it. A principal who suspects key theft must also revoke the device or

recover the key under TIP's existing transactions, and clients offer all three in one flow.

13.7 Stopping a harmful agent whose controller will not

If a registered agent is attacking people and its controller is unreachable or is the attacker, someone else must be able to stop it. AGENT_SUSPENDED may be signed by the controller, by an emergency panel the AI Trust Council appoints, acting by threshold, for up to 72 hours pending adjudication, or under a court or regulator's order recorded as a governance transaction. Sites treat a suspended agent as revoked for new actions.

13.8 Key events and outstanding mandates

Routine key events must not halt every agent at once. KEY_ROTATED carries `continuity`, either `routine` or `compromise`. After a routine rotation, mandates signed by the old key keep working until their own expiry, capped at seven days, and clients prompt the principal to re-sign them. After a compromise rotation, a device revocation or a key recovery, they stop at once. When a Verification Provider's accreditation is revoked and the revocation cascades to the TIP-IDs it verified, those TIP-IDs cannot issue new mandates, and existing mandates continue until a grace period the AI Trust Council sets ends, unless the cascade is marked as fraud, in which case they stop at once. Before any action that would end outstanding mandates, the client tells the principal how many will stop.

theailab.org

14. Agent-made content and Verifiable Assets

14.1 Who signs what an agent makes

When an agent creates content, the content needs a CTID like any other Verifiable Asset. A Verifiable Asset is any piece of digital content that carries a CTID (Content TIP ID): a permanent, signed public record of who made it, when, how it was made, and a fingerprint that shows whether it has changed since. (See theailab.org/verifiable-assets.)

The agent cannot sign as its principal, because the principal's key never leaves the principal's device and every signature with it needs a fresh biometric. The design adds a new attribution mode, `delegated`, alongside TIP's existing `self`, `employed` and `hosted` modes.

1. **A verb of its own.** The agent's mandate must include the `register_content` verb with the TIP ledger as its audience. An agent allowed to update a website is not thereby allowed to register content in its principal's name.
2. **Registered agents only.** Only registered agents can register content in this mode, because an ephemeral key has no public record to resolve.
3. **Checkable by every node, without publishing the chain.** The registration carries hashes of the mandate chain with a selective-disclosure proof that reveals only the `register_content` verb, the ledger audience, the validity window and the signatures, so every ledger node can check the authority at commit time and the ledger never stores the principal's business relationships or purposes.
4. **Honest attribution.** The CTID's author segment comes from the agent identifier, and readers see "registered by an agent acting for" the principal, not "created by" the principal.
5. **No race for priority.** A `delegated` registration never wins first-registration priority over a `self` registration of the same content, so an agent cannot register other people's work by the thousand to claim it first.

14.2 The origin floor

Content registered in `delegated` mode can never carry the OH (Original Human) origin code. The minimum is AA (AI-Assisted), and AG (AI-Generated) applies whenever the agent did the substantive creative work, which is the usual case. This follows TIP's conservative labeling rule and closes an obvious laundering route, in which AI-generated content is pushed through an agent to pick up a human label. A principal who wrote the content themselves registers it in `self` mode with their own key, as they do today.

theailab.org

14.3 Linking actions and content

The content registration carries the hash of the action request that produced it, and once the CTID is committed the agent or site can publish a short signed link from the receipt to the CTID. A reader of the content can find the action that produced it, and an auditor of the action can find the content it produced.

theailab.org

15. Protecting the agent's inputs

TIP cannot stop a model from being misled, but it can give agents better information about what they read.

15.1 Instruction provenance

Agents should treat instructions according to where they come from. Instructions from the principal arrive only through the principal's own signed channel. That channel is the principal's TIP client, or an interface that the agent's controller has bound to the principal's account at the agent and that authenticates the principal. A message that only claims to come from the principal, in an email or a document, is data. The mandate's `purpose` was written by the requester and is never treated as an instruction. Everything else, including web pages, emails, documents and other agents' outputs, is data, not commands. Where that data carries a CTID, the agent can see whether it was registered, by whom, with what origin code and at what trust tier, and agent frameworks can use that as one input in deciding how much weight to give it. Text that tries to grant the agent new authority is never authority, because authority comes only from a signed mandate.

15.2 Signed tool and server manifests

Tool poisoning hides instructions in tool descriptions, and "rug pulls" change a tool after a user approved it, because most clients do not compare tool definitions between sessions. Under this design, an MCP server or tool provider registers its tool manifest as a TIP content record signed by its TIP-ID. The agent pins the manifest's hash when the principal approves the tool. If the description changes, the hash changes, and the agent signer refuses to act through that tool until the principal approves the new version.

Skills, prompt packs and saved automations get the same treatment. A skill is a package of instructions, and often code, that an agent loads to do a kind of task, and it steers the agent as surely as a tool description does. The February 2026 marketplace incident in Section 2.2 showed infostealers shipped as skills. Under this design a skill is a Verifiable Asset. Its author registers it as a TIP content record, so it carries a CTID that shows who registered it, when, and whether it has changed since. The agent pins the SHAKE-256 digest of the skill's exact bytes when it is approved, and lists that digest and the CTID in the build statement (Section 7.7). The signer refuses to act with a changed or unpinned skill until it is approved again. The controller approves a skill, together with each principal whose mandate pins `builds`, since a new skill changes the build label for all of them. A skill that an agent writes is an instruction set that persists, so it is treated as `unverified` material (Section 15.4) until a person has reviewed and approved it. Where it was written under a work mandate, it is registered in `delegated` mode under that mandate, which needs the `register_content` verb with the TIP ledger as audience (Section 14.1). A CTID shows who made a

skill and whether it changed. It does not show that the skill is safe to run, so pinning limits what can change after approval and does not replace review before it.

15.3 Agent cards and manifests are claims

A2A Agent Cards and TIP agent manifests prove who published them, not that they are true. TIP adds two things a signed card lacks. The publisher is a verified TIP-ID, so a false card has an accountable author. False claims can be disputed through TIP's adjudication, and an upheld finding stays on the public record.

15.4 Agent memory

Many agents keep memory between tasks, as notes, summaries of past work, saved preferences and retrieved documents. Memory is an input like any other, with two differences. It persists, so one planted note can steer every later task. It also gathers in one place what the agent learned from many principals and sites, on a machine that someone else may run.

1. **Every entry records its source and its data classes.** A memory entry carries a source kind from the Memory Source registry and the data classes of the material it was written from. `principal` marks text the principal typed or sent through the principal's own channel. It never covers an attachment, a forwarded message or quoted material, which are `unverified`. `receipt` marks the fact that the agent took an action, with its digests and the receipt or TIP Agent Log entry. The content a site returned is `unverified`. `content` marks material that carries a CTID. `unverified` marks everything else, including web pages, emails, other agents' messages and the model's own summaries.
2. **The signer keeps the labels.** The runtime writes memory through the agent signer, which assigns the label and stores it where the model cannot change it. The label is the operator's assertion. Nothing outside the runtime can check it.
3. **Memory is data.** Section 15.1 applies to what an agent remembers as it does to what it reads. Only `principal` entries may be treated as instructions. A CTID shows who registered material, not that it is safe to obey, so `content` entries are data too. This is a rule for how a runtime presents memory to its model. It lowers the chance that planted text is obeyed. It does not stop a model from being misled.
4. **Memory that mattered is in the basis.** Where an action carries a basis digest, the memory entries the agent relied on are listed in the basis manifest (Section 10.6), each with its source kind.
5. **One partition per principal.** Memory gathered under one principal's mandates is kept apart from memory gathered under another's, and is used only for that principal's tasks. A principal here is the root issuer of the mandate and, where the mandate carries an end user as an opaque reference (Section 20, item 5), that end user. One company's customers are then kept apart from each other as well as from other companies.

6. **Encrypted at rest.** Each partition is encrypted under a key in the agent signer's custody. Stored partitions use AES-256-GCM. A partition key is wrapped to another party's key only with ML-KEM-768 (FIPS 203), never with a classical key exchange, and the wrapped key carries `kem_alg`. What that protects depends on the custody class. Under `tee` or `device_enclave`, with an attestation result (Section 7.8), and where the whole runtime that reads memory runs inside the measured environment, the party that hosts the machine cannot read the stored partitions. Under `software`, `hsm` and `cloud_kms` custody the host can read them, because the host can ask for the key. Encryption never protects memory from the agent's controller, or from a model provider that receives it in a prompt.
7. **Writes are logged, and loads are checked.** Each write adds a `memory_write` entry, with the source kind, the data classes and a digest, to the TIP Agent Log (Section 12.9). When a partition is loaded, the runtime checks it against those entries and against erasure entries, so an old copy restored by a host is noticed.
8. **Export and erasure.** A principal can have their partition exported (Section 7.9) or erased. Erasure follows Section 12.8, item 6.

TIP cannot see inside a runtime, so a site cannot check these rules. They are conformance requirements on the agent runtime (Section 23). An inspection tests them (Section 16.5), and a breach is a finding in a dispute (Section 16.2).

15.5 Flows between sites

Every check in Section 11 looks at one action at one site. Some harm appears only when steps are combined. An agent may read a customer file at one site and post a message at another. Each step is inside its mandate, and together they are a leak. The same weakness appears when the result of one step is taken as permission for the next.

1. **Results never raise authority.** The output of a tool, a search, a diagnostic or another agent is data. It never widens a mandate, waives a TIP Approval or changes a limit. Step-up is judged against the mandate and the site's own endpoint definitions, never against what an earlier step returned. A message that says "approved" is not an approval, because only a TIP Approval is one (Section 9).
2. **The `onward` constraint.** A mandate may say, for each data class, which audiences and inference places may receive material of that class once the agent has read it. The agent signer keeps a list of the data classes the agent has read for each principal, across every mandate that principal has issued to the agent, so a second mandate does not start with an empty list. A site declares the data class of each resource in its published endpoint definitions, as it declares the environment (Section 8.3). A resource with no declared class, and every response from a site that does not speak TIP, counts as every class in the mandate's `data_classes`. Reading a memory entry adds that entry's classes. Once a class is on the list, the signer refuses any request to an audience, and any `infer` to a place, that the constraint does not list for that

class. The rule covers the whole request, including the address, the query and the headers. The list lasts as long as that principal's memory partition (Section 15.4), not one task or one mandate. Material of a class that `onward` does not name may go to no audience and no inference place, except material of the class `public`. Each decision is a `flow` entry in the TIP Agent Log.

3. `task_ref`. The signer creates a random `task_ref` and gives it to every request in one task, covered by the signature and copied into each receipt (Section 10.1). A reviewer can then gather a task's receipts across sites and read the whole path. It carries no authority, and no limit depends on it, because an agent that wanted to dodge a limit could start a new task. The count of Section 15.6, item 7, uses the signer's own task identifier, never `task_ref`. Two sites that compare receipts can see that one task touched both, so an operator that needs its sites kept apart sets its signer to leave it out.
4. **Totals stay with mandates.** Limits that must hold across a sequence, such as `amount_total`, `max_uses` and `step_up_total`, stay attached to mandates and counters (Section 8.5), which the agent cannot reset.

The `onward` rule stops accidental flows and simple attacks. It does not stop a determined attacker who has hijacked the model, even when the signer is intact. Material can leave through an audience the constraint lists that many unrelated people use, such as a mail service or a code host. It can pass to a sub-agent, whose signer keeps a list of its own. It can be shown to the principal in a form that a chat program then fetches from an outside address, or leave through a model provider's own tools (Section 18.2). The rule tracks classes of data and cannot judge meaning. An attacker who takes over the agent's host (T38) can go around it altogether.

15.6 Output the signer cannot read

An agent may begin to emit material in a form that neither its principal nor the receiving site anticipated. Examples include source code in a field that expects prose, base64 or another binary-to-text encoding, ciphertext, a compressed archive, a natural language the principal does not use, and a notation that two agents have developed between themselves. These forms have legitimate uses, and they also prevent review, because neither the principal nor a later reviewer can tell what the material says. They do not by themselves defeat the `onward` rule of Section 15.5, which follows the data classes the agent has read and not the appearance of what it sends. Unreadable output is, however, the form in which material that the signer's list does not show would leave unseen. Such material includes what reached the model before a read was logged and what the signer could not attribute to a source, and item 4 treats unreadable output more strictly for that reason. A change of form is a signal that warrants review. It is not evidence of misuse, and the rules below do not treat it as evidence.

The rules rest on Section 15.1. Material from another agent is data whatever its form. It confers no authority, cannot widen a mandate and cannot stand in for a TIP Approval, so two agents that

converse in an opaque notation gain no permission by doing so. The rules below govern only where such material may go, what is recorded about it and who reviews it.

1. **Format entries.** A format entry is an object with `media_type`, an optional `profile` and an optional `languages`. `media_type` names an entry in the Format registry (Section 6.5), in lowercase, with no parameters, wildcards or structured syntax suffixes, so `application/json` and `application/ld+json` are different entries. `profile` names an envelope schema registered under that media type, such as a common shape of request to a language model. A profile describes a structure and never names a provider, and the registry admits a profile only where more than one provider uses its shape, so a list of formats reveals no other party (Section 8.6). `languages` lists language tags under BCP 47 (RFC 5646), compared by the basic filtering of RFC 4647, so `en` covers `en-US`. Each registry entry names a strict parser and states whether the content of its format can be inspected, which parts carry free text, such as the string values of a JSON document, and which encodings a signer looks for inside them. Text formats are UTF-8 without a byte order mark, and text in any other character encoding is unreadable under item 4. The registry also holds one entry, `http-request`, whose detection rules apply to the target URI, the query and the headers. It is not a media type and never appears in a format entry. The first registry entries are specified with the conformance suite, and Section 23.2, item 12, says what applies until then.
2. **Declared formats.** A site may list, in each entry of its endpoint definitions (Section 10.7), the format entries that endpoint accepts. A site that declares formats checks the body at check 11 of Section 11, after it has confirmed that the body hashes to `content_digest`. It first decodes any content coding that the registry lists, within the bounds of Section 6.6. The body matches only when both the registry's parser for a declared entry and the parser that will carry out the request accept it, and the site otherwise refuses the request with the reason code `format_not_allowed`. Under the multipart envelope of Section 10.2 the check applies to the part that carries the API request, and a request with no body passes it. A site that declares no formats accepts any body its own parser accepts, as it would without this section.
3. **The `formats` constraint.** A mandate may carry `formats`, a list of format entries the agent may send under that mandate to any audience or inference place (Section 8.3). The agent signer checks the body of each request made under the mandate and refuses one that does not match a listed entry. For an `infer` request the entry's `profile` locates the material inside the request envelope, and the signer checks that material as well as the envelope. A listed entry without `profile` never matches an `infer` request, so a mandate that limits the formats of inference lists a profile. A child mandate's list lies inside its parent's entry by entry. A child entry lies inside a parent entry that has the same `media_type` and `profile` and either carries no `languages` or carries `languages` that cover every tag of the child entry. A child entry without `languages` lies only inside a parent entry without `languages`. A mandate without `formats` imposes no limit beyond those of the site, and `formats` never applies to traffic that names no mandate. Like `inference` and `onward`, `formats` is the operator's undertaking. A site checks only

that it narrows correctly, and the rules of Sections 8.3 and 11 on ephemeral keys with no registered agent above them apply to it as they apply to `onward`.

4. **Unreadable material.** Material is unreadable when the signer cannot determine its format, when the registry marks its format as not inspectable, as with ciphertext or an archive, or when the signer cannot open it within the parsing bounds of Section 6.6. A string, part or field inside an inspectable format is unreadable in the same way when the registry's detection rules mark it as encoded, compressed or enciphered, as with base64 inside a JSON string or a `data:` URI inside HTML. A nested encoding whose decoded content matches a format entry the mandate lists, in a format the registry marks inspectable, is checked as that format and is not unreadable, so an image sent to a model under a mandate that lists its type is checked as an image. The signer applies this item to the body, and to the target URI, the query and the headers under the `http-request` entry, because the `onward` rule covers the whole request. A request body under a content coding that the registry lists is decoded within the bounds of Section 6.6 before it is checked.
   - **Fields the signer writes.** Item 4 does not apply to the header fields `Signature`, `Signature-Input`, `Signature-Agent` and `Content-Digest`. Nor does it apply to `tip-audience`, `tip-mandate-ref`, `tip-intent`, `tip-content-digest`, `tip-basis-digest`, `tip-build-digest`, `tip-agent-log-head`, `tip-task-ref` and `tip-session`. Nor does it apply to the parts of a Section 10.2 envelope or preflight that carry `chain`, `keys`, `records`, `attestation`, `inspection`, stapled status and approvals. The signer builds these from its own state, and they carry no material the agent read. Item 4 does apply to the target URI, to `tip-action` and `tip-resource`, to `idempotency-key` and to the part that carries the API request.
   - **Credentials.** A credential header is exempt only when the signer, or the credential broker of Section 7.10, item 1, inserted its value from its own store. Every other credential header, including one that the agent writes for a site that does not speak TIP (Section 12.9, item 3), is checked like any other header. A bearer token or cookie that the agent writes for such a site is therefore usually unreadable, so an agent that must call the site after reading a listed class needs `onward` to list that site for every class, or a credential broker that inserts the credential.
   - **Classes.** Unreadable material is treated as carrying every class in the mandate's `data_classes` and every class on the signer's list for that principal (Section 15.5), and `personal` where the mandate carries no `data_classes`. It may therefore leave only for a place that `onward` lists for every one of those classes, and for no place when `onward` omits any of them. Unreadable material in a request that names no mandate carries the union of those classes across every mandate and list loaded in the task.

5. **Language.** Language tags are advisory in version 1. Language identification is unreliable on short strings, names, code and mixed text, and a wrong result would refuse honest work. A signer records a mismatch between the language it detects and the listed tags as a `format_anomaly` entry, and never refuses a request or restricts a mandate on language alone.

6. **Record every case.** The signer writes a `format_anomaly` entry to the TIP Agent Log (Section 12.9) whenever it refuses a request under item 3, whenever item 4 finds unreadable material, whether the request then leaves or is refused, and whenever item 5 finds a mismatch. A mandate that carries neither `formats` nor `onward` refuses nothing under this section for requests made under it, and still leaves a record of each piece of unreadable material sent under it. A request that names no mandate is recorded against the mandates that item 7 names, or that it would name were the request refused, and with `mandate_id` `none` where there are none.
7. **Restrict after repeated refusals.** A single refusal under item 3, or under the `onward` rule as item 4 applies it, restricts nothing. Any page, message or agent the model reads can ask it to answer in base64, and if one refusal restricted a mandate, anyone able to place text before the agent could freeze its mandates at will.
   - **Counted by task.** The signer gives each unit of work an internal task identifier when the runtime starts it, keeps that identifier whether or not it sends `task_ref`, and lets no other component set it. Refusals within one task count once in each window of Section 6.6, because an injected instruction stays in the model's context and a model that retries it within seconds shows one cause, not several. Every tenth refusal within one task in one window counts again, so a model cannot probe encodings without limit inside a single task.
   - **Threshold.** When the refusals counted against one mandate reach the threshold that its `restrict_after.format` sets, or where it sets none the default in Section 6.6 of three within 10 minutes, and never fewer than two, the signer restricts that mandate. It then refuses every further request under it except declared safe-state actions (Section 13.5).
   - **Requests that name no mandate.** A refusal of such a request counts against every loaded mandate whose `onward` rule would refuse it, and never against a mandate that carries no `onward`.
   - **Ending a restriction.** A restricted mandate stays restricted until it expires, or until the principal ends it with a stop notice (Section 23.2) or, once the ledger transactions are live, a revocation. The principal may sign a replacement mandate, which is a new mandate, starts unrestricted, may carry the same scope and may list the refused format. The signer never stops or revokes a mandate. It restricts one, and only the principal can end the restriction.
8. **Show the principal, safely.** The principal's TIP client presents each refusal and each restriction with the destination and the format the signer detected. It also lists the inputs the agent read earlier in the same task, each with its source kind and the origin or address it came from, so that the principal can see where an instruction to change form may have come from. The signer keeps the refused bytes in an evidence store that is separate from every memory partition, encrypted under Section 15.4, item 6. The store is never presented to the model and holds no memory entries, and the operator keeps no other copy. The bytes are kept for the period that Section 6.6 sets and erased under Section 12.8, item 6. The client shows the bytes to a principal only where every input the agent read in that task was read under that principal's mandates or from that principal's memory partition, and any other principal sees the entry without them.

theailab.org

Material read from a site that does not speak TIP names no mandate, so where such a read is the likely source of an injected instruction, no client shows the bytes, and they stay available only under Section 12.8, item 5. It shows them first as inert text, with control and bidirectional characters escaped, and decodes them only within the parsing bounds of Section 6.6, with no script, no active content and no network fetch. Outside the principal's own client, the bytes go only where Section 12.8, item 5, allows.

These rules act on form. They cannot detect meaning carried inside material of an allowed form, such as a phrase agreed in advance, a choice among synonyms, the order of items in a list, the timing of requests or the length of a field. Ziegler, Deng and Rush reported in 2019 that text in which a language model had hidden a message was judged realistic by human evaluators, and no parser that admits ordinary prose excludes that channel. Parsers can also disagree. A file built to parse as two formats, or an encoding that the registry's detection rules do not list, passes a check that it ought to fail. A conforming signer bounds the size, expansion and nesting depth of what it parses (Section 6.6) and treats material beyond the bound as unreadable under item 4. A party that can start tasks for the agent, such as a sender of email to an agent that triages a mailbox, can still bring a mandate to its threshold and restrict it, at the cost of one task for each counted refusal. A restriction applies to one mandate, so an agent that holds a sibling mandate from the same principal, or a sub-agent that holds an earlier sub-mandate, can still send under those until the principal ends them. A host that bypasses the signer altogether evades every rule here (T38). The protections that remain in each of these cases do not depend on reading the material. Material confers no authority (Section 15.1), mandates can be kept narrow, and high-risk actions need a TIP Approval or, under the interim profile, are refused.

theailab.org

16. Accountability and reputation

16.1 Accountability stays with people and organizations

An agent has no trust score of its own. Scores belong to TIP-IDs, and every TIP-ID belongs to one verified person or organization. This is deliberate. If agents had independent reputations, a controller with a ruined agent would register a fresh one. Because conduct traces to the controller and the principal, and deduplication makes a second TIP-ID hard to obtain, there is no fresh start to buy.

16.2 Disputes about agent conduct

A receiving site, a principal or an affected third party can open a dispute about an agent's conduct, attaching receipts and inclusion proofs as evidence. Disputes run through TIP's existing three-stage pipeline. The kinds of finding the RFC should define include a false manifest, a false basis digest (Section 10.6), a false build statement (Section 7.7), a TIP Agent Log that omits actions (Section 12.9), a false inspection record (Section 16.5), conduct outside a mandate that the controller's own signer should have refused, and a controller ignoring revocation. Any score effect on a controller or principal TIP-ID must go through a SCORE_UPDATE transaction with a stated reason, under TIP's single-channel rule. The size of those effects is a decision for the AI Trust Council, and this paper proposes none. Disputes whose evidence contains regulated data, such as health records, controlled government data or customer financial data, and disputes that rely on a basis manifest (Section 10.6), never go to anonymous community jurors. They go to a confidential panel of accredited experts bound by confidentiality, and only the finding is published. Organization TIP-IDs are linked to the personal TIP-IDs of their beneficial owners, and conduct findings attach to those owners as well, so a fresh shell company is not a fresh start.

16.3 What refusals do not mean

A refusal receipt is not evidence of misconduct by itself. Sites refuse for many honest reasons, and a hostile site could refuse valid requests on purpose. Refusal counts are shown for context and never feed a score. What a refusal does prove is that a specific, signed request was made, because a site cannot forge an agent's signature on a request the agent never sent.

16.4 Conduct signals for sites

Sites still want to know whether an agent behaves. The public record supports that without a new score. A registered agent's age, its controller's TIP-ID and trust tier, any certification of the AI system behind it, and its upheld disputes are all visible. Sites combine those with their own experience of the agent.

theailab.org

16.5 TIP Inspection

Everything an agent's operator says about its own runtime is a statement by an interested party. That covers the manifest, the build statement, how memory is kept and whether the log is complete. A vendor's badge adds nothing to that. An examination by an accountable party that the principal accepts adds a second signature and a record of what was tested.

1. **Who.** An inspector is an organization with an active TIP-ID. A mandate that requires an inspection names the inspectors it accepts, so the choice of inspector belongs to the principal and not to the party being inspected. An inspector should not be owned or controlled by the agent's controller. A site cannot check that, which is why the principal names the inspector. Open question 16 in Section 26 asks who accredits inspectors.
2. **Under a mandate.** The controller issues the inspector's tooling a mandate with the verb `inspect`, signed under Section 8.2 like any mandate, which is the controller's approval of the inspection. Its resources are the scope of the inspection, and it lasts no longer than the limit in Section 6.6. `inspect` only reads. Tests that act, such as those in item 3 that try to reach a site without the signer, run under a separate test mandate that the controller signs for a test site the inspector operates, never under a principal's mandate. The agent signer checks each of the inspector's signed requests against that mandate and writes an `inspection_access` entry in the TIP Agent Log. The inspector keeps its own record of what it asked and what it was shown.
3. **What is examined.** The inspector checks the running build against the build statement (Section 7.7). It checks attestation results and signer states (Section 7.8). It checks that the signer or its policy proxy is the agent's only way out, and that the agent holds no ordinary credential for a site that accepts TIP (Section 7.4). It checks how memory is kept (Section 15.4), the TIP Agent Log and its coverage entries (Section 12.9), and pinned tools and skills (Section 15.2). It runs the conformance tests for the Agent Signer and Agent Runtime classes (Section 23), and the tests of the Containment class from inside the boundary (Section 7.10, item 10). It marks the isolation class `measured` only where it ran those tests itself. It also runs tests that the operator is not told of in advance, such as an attempt to reach a site without the signer and an attempt to act without a log entry. It reads configuration, digests and logs. It reads the contents of memory or of requests only with the agreement of the principal they belong to.
4. **Measured or reported.** A runtime that knows it is being inspected can show a clean face. The inspector therefore marks each finding as `measured`, where it rests on an attestation result the inspector checked itself or on a test it ran itself, from outside the runtime or, for the Containment class, from inside the boundary, or as `reported`, where it rests on what the runtime said or on a test it only watched.
5. **The record.** The inspector signs an inspection record, an object with `typ` `tip-inspection/1`. It names the registered agent, by identifier or, under an interim agent record, by key thumbprint. It gives the `build_digest` examined, the scope of the `inspect` mandate it worked under, each conformance class tested with its result and its findings, the date and an expiry no later than

the validity in Section 6.6. An inspector registers a failing record as it does a passing one, so a controller cannot collect inspections and show only the one that passed. The inspector registers the record as a TIP content record, so it has a CTID. An inspector withdraws a record by retracting that content record, and an upheld dispute finding against a record withdraws it as well.

6. **Requiring one.** A mandate may carry the `inspection` constraint (Section 8.3). The agent then staples the record in the session preflight of Section 10.2. The site checks that the inspector is one the mandate names and that the signature is valid. It checks that the record names the acting agent and the `build_digest` in the request, and that every required class passed. It checks that the record is within the allowed age, unexpired and not withdrawn, from ledger state no older than the smallest `max_status_age` in the chain. It refuses with `inspection_required` otherwise. The receipt carries `inspection_seen`, so a later verifier can find the record and see whether it had been withdrawn. A new build label needs a new inspection. TIP clients refuse to sign an `inspection` constraint that names no inspector.

A claim of conformance for the Agent Signer and Agent Runtime classes says which of three stages applies. **Self-declared** means the operator ran the conformance tests itself. **Inspected** means a TIP Inspection record covers the current build label. **Log monitored** means, in addition, that monitors check the agent's anchored seals for forks and for late anchoring (Section 12.3). A monitor sees seals and cannot read entries, so this stage says nothing about whether the log is complete.

An inspection is the inspector's signed opinion at one moment about one build label. In the terms of Section 12.10 it is `stated`. It is not proof that the agent is safe, or that the runtime behaved the same way when nobody was looking. An inspector that signs a false record has made its own signed misstatement, and that is a finding under Section 16.2.

theailab.org

17. Threat model

The table lists the attacks the design has to stop, how it stops them, and what remains. “Residual” is stated plainly so nobody mistakes a narrowed risk for a closed one.

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T1	Impersonation	Agent claims to be a known agent	Request must verify against the registered agent key or the mandate-named key	Depends on the integrity of the ledger copy the verifier uses and on key safety
T2	Forged authority	Company A signs a mandate in company B’s name	Root signature must verify against B’s registered TIP-ID key	None
T3	Stolen mandate	Thief replays a mandate with their own key	Mandate names one agent key or registered agent; thief’s key is not it	Where the mandate names a registered agent without a pinned key, a rotation by a compromised controller changes the key; see T41
T4	Stolen agent key	Thief signs requests as the agent	Keys in HSM, KMS, TEE or enclave; short mandates; AGENT_REVOKED; MANDATE_EPOCH_ADVANCED	Works until revoked or expired; short lifetimes bound the window
T5	Replay	Same request sent twice	Nonce plus short lifetime	None
T6	Cross-site replay	Request for site A replayed at site B	`audience` inside the signature	None
T7	Scope escalation	Agent acts outside its mandate	Agent signer refuses; site refuses	Sites that skip checks or map endpoints to the wrong verb; conformance testing (Section 23)
T8	Escalating sub-delegation	Agent grants a helper more than it holds	Narrowing rules on every axis; segment-aware prefixes; no regex	None
T9	Chain loops and depth bombs	Circular or very deep chains to exhaust verifiers	Parent and root links; `max_depth` ; hard maximum chain length	None
T10	Confused deputy	Agent tricked into using its authority for someone else	Mandate only narrows the root issuer’s rights at the site; agent and controller sessions never authorize mandate actions (Section 8.7)	Within-scope misuse remains possible; see T21

thevlab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T11	Body tampering	Agent signs +100, -100 is delivered	Exact-bytes SHAKE-256 digest, never the normalized content hash	None
T12	Time-of-check versus time-of-use	Site checks one body and executes another	Digest checked on the bytes executed; `result_digest` in receipt	Site insider; see T16
T13	Receipt forgery or edit	Receipt changed after the fact	Site signature plus log inclusion against a witnessed, anchored TIP Seal; receipt keys in hardware with revocation (Section 12.7)	A stolen receipt key allows forged new receipts until it is revoked
T14	Log fork	Site shows different histories to different auditors	Witness quorum and consistency proofs; ledger anchoring	Needs collusion of the site and the witnesses that two majorities share, which is one witness in a set of three (Section 12.10)
T15	Receipt withholding	Site acts but logs nothing	Synchronous receipts; agent-side logs; anchored log sizes	Detectable, not preventable
T16	Site lies about authority	Site logs an action the chain did not allow	Verifier re-derives authority from the signed chain, never trusts the site	None for detection
T17	Clock manipulation	Site backdates or postdates a decision	Bounds from request `created` and ledger anchor time	Uncertainty equal to the anchor interval
T18	Revocation race	Agent acts in the gap after revocation	Short `max_status_age`; status pushed by nodes; step-up actions check status live	Window equal to `max_status_age` for non-step-up actions
T19	Status denial	Attacker blocks status checks hoping sites fail open	Fail closed for ordinary changes; threshold status from several independent signers; safe-state actions stay available (Section 13.5)	Ordinary actions stop during a long outage; safe-state actions continue with degraded receipts
T20	Algorithm downgrade	Labels stripped or swapped to a weak scheme	Labels inside the signed body; verifier allow-list	None
T21	Prompt injection within scope	Hijacked agent does something allowed but unwanted	Narrow mandates; step-up for irreversible actions; instruction provenance; receipts for reversal and dispute	Real; limited to the mandate's scope

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T22	Tool rug pull	Tool description changed after approval	Pinned manifest hash; signer refuses changed tools	None once pinning is in place
T23	Lying manifest or agent card	Agent describes itself falsely	Verified publisher; adjudication; public findings	Lie possible until disputed
T24	Sybil agents	Bad controller spins up new agents to escape history	No agent scores; conduct traces to a deduplicated TIP-ID; per-controller registration limits	Bounded by TIP-ID deduplication strength
T25	Mandate phishing	Fake service asks a person to sign a broad mandate	TIP client renders the mandate and shows the requester's verified TIP-ID and domain; ceilings on duration and verbs	Social engineering of a careless signer
T26	Mandate harvesting	Malicious site collects mandates it was shown	Mandates are useless without the agent key; audience binding; per-site mandates and selective disclosure	The harvested mandates still reveal business details the site was shown
T27	Controller compromise	Attacker controls an organization's signing member	Roster quorum for high-risk mandates; security-contact revocation; epoch advance	Window until detected
T28	Model holds the key	Injected prompt makes the model print its key	Key held by the agent signer, never in model context	None if custody rules are followed
T29	Verification flood	Junk requests to exhaust site CPU	Cheap checks before signature checks; rate limits per agent and per controller	Ordinary denial-of-service risk
T30	Canonicalization tricks	Two encodings of one mandate, or hidden fields	TIP canonical JSON; reject duplicate keys and unknown fields	None
T31	Counter evasion	Agent spreads spending across many sites	Per-site counters by root; counter service for global limits	Global limits need single-site mandates or a counter service
T32	Human-label laundering	AI output pushed through an agent to look human	Origin floor of AA in `delegated` mode	A principal can still sign AI output in `self` mode; detection rests on the classifier and disputes
T33	Privacy harvesting	Observer profiles a company from public records	Only hashes and roots on the ledger; receipts held by parties; selective disclosure	Agent registration itself is public

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T34	Quantum attack on old records	Future quantum computer forges old signatures	ML-DSA-65 and SHAKE-256; algorithm labels allow stronger schemes to be added later	Depends on the post-quantum assumptions NIST standardized
T35	Verification Provider compromise	A VP issues fraudulent TIP-IDs	TIP's existing REVOKE_VP cascade and accreditation controls	As in TIP today
T36	Backdated mandate	Holder of a retired or stolen key signs a mandate dated before the rotation	New actions need the key in effect at decision time; future `issued_at` refused	None for new actions
T37	Mislabeled action	Agent labels a delete as an update to fit its mandate	Site derives the action from its own endpoint definitions and refuses mismatches	None where endpoints are declared correctly
T38	Agent host compromise	Attacker takes over the machine running the agent signer	Signer enforces the mandate; hardware custody stops key export; step-up blocks high-risk actions; revocation	Attacker can act within scope until revoked
T39	Step-up swap	Approval for one action replayed on another, such as an empty-bodied delete	Site-signed challenge binding method, URI, audience, action, resource, digest, display and mandate; consumed atomically	None
T40	Approval fatigue	Flood of prompts until the principal taps approve	Rate limits; denials suspend the mandate; deny-and-stop in every prompt	A tired person can still approve
T41	Vendor key swap	Breached controller rotates every agent to its own keys	New key must sign rotations; notice delay; pinned keys on step-up mandates	Window of the notice delay, with the principal told
T42	TIP Stop All dodge	Thief signs mandates with a future date to survive an epoch change	Integer `issuer_epoch` raised only on the ledger	A thief who still holds the issuer key can sign new mandates at the new epoch; key revocation is required (Section 13.6)
T43	Junk-request amplification	Invented mandates naming throwaway keys force signed refusals	No signed receipt until the full chain verifies to an active TIP-ID; signed refusals rate-limited per root issuer	Ordinary denial-of-service risk

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T44	Chain-URL abuse	Fetch URL points at the site's own network	Fetch only after the chain signature check; resolve the host once, check the address is public, connect to that exact address, follow no redirects, cap at 64 KB and 2 seconds	None where resolution is pinned
T45	Path tricks	`..`, encoded slashes or case variants slip past prefix checks	RFC 3986 normalization; dot segments and encoded slashes refused	None where the router and matcher share normalization
T46	Status-field widening	Sub-mandate sets a long status age or a dead status list	Status age narrows down the chain; each link checked against its own entry	None
T47	Cross-site tracking	Sites compare a personal principal's TIP-ID	Organization principals for consumer services; agent-only profile for low-risk reads (Section 20)	Real for personal principals using full chains
T48	Stolen TIP-ID key rotated	Thief with a principal's key signs a key rotation to take over	Hardware-locked keys that never leave the device; see open question 8	A core TIP question, beyond this paper
T49	Mandate beyond the issuer's own rights	Person with no rights at a site signs a mandate for an agent that has access there	Mandate only narrows the root issuer's own rights; site authorizes against the linked account (Section 8.7)	None where account linking is correct
T50	Agent approves for its human	Browser or computer-use agent opens the approval prompt and clicks Approve	Approval on a device the agent cannot drive; no password approvals for high-risk verbs (Section 9.2)	A person who approves without reading
T51	Hidden amount or dose	Approval screen shows a path, not the amount, payee or dose	Site-signed typed display entries required per action class (Section 9.1)	None for registered display types
T52	Double use of one approval	Two copies of one approval sent at once	Challenge consumed in the same atomic step as the nonce and counters	None

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T53	Rogue status source	Attacker runs a node that vouches for a revoked mandate	Threshold of Council-registered status signers; versioned, anchored status lists	Collusion of a threshold of status signers
T54	Legacy credential bypass	Agent uses an ordinary API key and never touches TIP	`tip_required` endpoints; egress through the signer (Section 7.4)	Sites that still accept legacy tokens for agents
T55	Structuring	A large payment split into many small ones under the step-up amount	`step_up_total` over a window	Splitting across sites without a counter service
T56	Reconnaissance from the registry	Attacker reads agent manifests to find vulnerable versions	No versions in public manifests; private visibility (Section 7.5)	Agent existence is visible for public agents
T57	Weaponized stop	Compromised security contact stops every agent during an incident	Single-contact stops only suspend; scoped stops; quorum restore (Section 13.6)	Up to 72 hours of suspended agents if the quorum does not act
T58	Status or coordinator outage	Central ledger component or status service unavailable	Several independent status signers; safe-state actions; degraded receipts (Section 13.5)	Ordinary actions wait for recovery
T59	Signature reuse across object types	A signature made for one object presented as another	`typ` in every object and FIPS 204 context strings (Section 6.4)	None
T60	Lost or altered original	A party loses or edits the record behind a digest, so an action cannot be proven or is misrepresented	An altered original fails to match its signed digest; parties keep originals at least as long as the receipt (Section 12.8)	A lost original cannot be recovered, and the digest then proves only that something was signed
T61	False basis	An operator commits to a basis the agent did not use, to shift blame or hide an input	`basis_digest` is covered by the agent's signature and fixed at the time of the action; disputes (Section 16.2)	TIP cannot prove what a model read or relied on, and a compromised agent host (T38) can feed the signer a false basis that is attributed to the operator

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T62	Hosted provider breach or misuse	A provider running a party's log, gateway or evidence store reads, leaks, alters or deletes what it holds, or uses the receipt key it holds to forge receipts and challenges	Hosting is optional; contract, residency and access-logging terms; encryption at rest; receipt keys in declared hardware custody with revocation (Section 12.7); altered originals fail to match their digests; the provider may not witness what it hosts (Sections 12.3 and 20)	Disclosure or loss of what the provider held, and forged receipts until the key is revoked
T63	Memory poisoning	A planted note in the agent's memory steers a later action	Source labels kept by the signer; only `principal` entries may be presented as instructions; writes logged; memory relied on listed in the basis (Section 15.4)	Labels are the operator's assertion and a model can still be misled; limited to the mandate's scope
T64	Memory read by the host	The machine or provider running the agent reads what it has remembered	Partitions encrypted under keys in the signer's custody; `custody` and `attestation_max_age` constraints (Sections 15.4 and 7.8)	No protection under `software`, `hsm` or `cloud_kms` custody, none against the controller, and none for memory sent to a model
T65	False build label	The model, agent software or a skill is changed and the old label is kept, or the label is changed without cause	Signer derives the statement from what it loaded; `build_digest` in receipts; `builds` refuses an unlisted label; inspection (Sections 7.7 and 16.5)	The label is the operator's claim unless an attestation result names the signer that computed it; a provider can change weights behind one version name
T66	Counterfeit or stale attestation	Attacker's own code in genuine hardware vouches for a key held elsewhere, or old evidence is replayed	Attestation verifier checks the measurement against reference values; result binds key, build label and a recent ledger transaction; `attestation_max_age` (Section 7.8)	As strong as the hardware maker and the verifier; shows nothing about compromise after measurement or software outside the measured part
T67	Unreceipted action	The agent acts at a site that issues no receipt, and no evidence exists	TIP Agent Log written at the signer and its policy proxy; no record, no release (Section 12.9)	One-party evidence; no mandate authorizes or checks the action at such a site, and `reach` limits only where the agent connects (Section 7.10); an attacker on the agent's host (T38) can act outside the signer

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T68	TIP Agent Log omission or rewrite	The operator leaves actions out, rewrites entries before the next seal, or shows a principal only part	`agent_log_head` in every receipt fixes earlier entries; witnessed and anchored seals; coverage entries checked against receipts the site gives the root issuer; dispute finding (Sections 12.9 and 16.2)	Entries for sites that keep no record rest on the operator until sealed, and an action never logged at such a site cannot be shown
T69	Harmful composition	Steps that are each allowed combine into a result nobody approved, or data read at one site is sent to another	Results never raise authority; the `onward` constraint over the whole request and for the life of the principal's memory partition; `task_ref` for review (Section 15.5)	Stops accidental and simple flows only; shared audiences, sub-agents, rendered output and provider tools remain; enforced by the signer alone
T70	Inference leak	Material is sent to a model provider the principal never approved	The `inference`, `data_classes` and `onward` constraints; a model gateway acting as receiving site (Section 18.2)	Enforced by the signer; `on_device` is the operator's assertion; what a provider does with material it lawfully receives is a matter of contract
T71	Session riding	An agent with no key of its own acts inside a person's logged-in session and is taken for the person	A site may ask the account holder for a TIP Approval on a separate device before a minimum-list verb, whoever appears to be asking (Section 18.3)	A site that does not ask cannot tell the difference; reads and lesser writes leave no sign; content the person then registers as their own is governed only by their duty to label it honestly (T32)
T72	Undeclared agent	Agents work for an organization outside its inventory	Work systems require TIP of agents that identify themselves; the organization compares declared agents with what its gateways observe (Section 18.3)	Any agent that uses a human login is unseen wherever people also log in without a TIP Approval
T73	Employer overreach	An organization reaches into the personal side of an employee's own agent	A separate work agent with its own key and build; memory partitions; log entries visible only under the organization's own mandates; scoped inspections (Sections 18.3, 15.4, 12.9 and 16.5)	Admission terms let the employer choose the work agent's build and hardware and see when it works; pressure through employment terms is outside the protocol

thetilab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T74	Coerced approver	A person is forced to approve with their own device	Dual control and separation of duties for organizations; review after break-glass; revocation and disputes afterward (Section 9.4)	A single approver acting under threat produces a valid approval, and TIP cannot see duress
T75	Poisoned release	A trusted update of the agent software, the signer or a TIP client carries the attack	Skills and tool manifests registered and pinned; a new build changes the label; `builds` and inspection (Sections 7.7, 15.2 and 16.5)	A poisoned signer reports any label it likes, and a poisoned signer or TIP client defeats every control that runs in it; a release poisoned before signing is trusted until found
T76	Transfer swap or lying source	An agent's state is altered before or while it moves, and the controller signs for it	Transfer statement over the state digest and the final anchored seal; destination checks before start; principals re-sign pinned mandates after their client checks the statement (Section 7.9)	The statement covers the state the source presented; the source can keep what it could already read and any credentials not yet rotated
T77	Inspector abuse	An inspector uses its access to read private data or to plant instructions	`inspect` only reads, under a short mandate the controller signs; contents need the principal's agreement; every access logged (Section 16.5)	The inspector sees the opened build statement, log metadata in scope and the configuration, and its tooling is one more input to the agent
T78	False, captive or borrowed inspection	A friendly inspector vouches without examining, or an agent staples another agent's record or one that failed	The mandate names the inspectors; the site checks the agent named, the build label, the result of each class and withdrawal (Section 16.5)	An inspector the principal trusted can still be wrong or bought; no accreditation exists yet (open question 16)
T79	Inspection facade	The runtime detects the inspection and shows a clean state	Findings marked `measured` or `reported`; tests the operator is not told of; attestation results the inspector checks itself (Section 16.5)	`reported` findings stay the runtime's own word

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T80	Hostile or breached model provider or gateway	The model's output steers the agent, a gateway alters prompts or answers, a different model answers, or material leaves through the provider's own tools	Mandate scope and step-up limit what a steered agent can do; `inference` names providers; gateways name the provider behind them; provider tools off unless listed (Section 18.2)	Full influence over the agent's reasoning inside its mandate; model substitution cannot be seen from outside the provider
T81	Credentials outside TIP	The agent holds ordinary credentials for sites that do not speak TIP, and a thief or a hijacked model uses them	Version 1 keeps those credentials out of the signer and out of the agent; a credential broker outside the boundary inserts each one for one destination and one principal (Section 7.10); traffic to those sites is recorded at the policy proxy, and credentials are rotated on departure (Sections 12.9 and 18.3)	Those credentials work outside any mandate until the site adopts TIP or rotates them; the broker gathers them in one place and is a target; the proxy sees request content in the clear
T82	Agent log key theft	A thief seals false entries or creates a fork that appears to convict the operator	Log key in the agent key's custody class; rotation and revocation with `compromised_since` (Sections 12.9 and 12.7)	False entries stand until the key is revoked
T83	Memory label laundering	Attacker text gains the `principal`, `receipt` or `content` label	`principal` covers only what the principal typed; returned content is `unverified`; `content` is data; the signer keeps labels (Section 15.4)	A principal who pastes hostile text has typed it; a compromised runtime writes any label
T84	Memory rollback	A host restores an old copy of a partition, undoing an erasure or removing notes	Partition checked against `memory_write` and erasure entries when loaded (Section 15.4)	Under `software` custody the host can also alter the check
T85	Self-written skill	A hijacked agent writes a skill that steers every later task	Agent-written skills are `unverified` until a person approves them; pinning; a new skill changes the build label (Section 15.2)	A reviewer who approves without reading

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T86	Unlock replay	A captured unlock is used again, or used to revive a source after a transfer	Unlock is single use, bound to the log head and a value from the signer, and refused once the rotation of a transfer has taken effect (Section 7.8)	A controller who signs a bad unlock
T87	Forced restriction	An attacker blocks the log store or the witnesses so that agents stop ordinary work, or sends safe-state actions at a harmful moment	Durable local append; safe-state actions stay available and are logged late with both times (Sections 12.9 and 7.8)	Ordinary work stops until the log returns; a safe-state action can itself be untimely
T88	Unchecked destination and overlap	The new host skips its checks, or the source keeps acting after the move	Principals re-sign pinned mandates only after their client checks the statement; the old key stops at the rotation; final seal anchored before the move (Section 7.9)	Unpinned mandates carry over unchecked; a kept `software` or ephemeral key works until sites refresh, up to `max_status_age`
T89	Custody downgrade by transfer	An agent and its memory move from hardware custody to `software` custody	AGENT_UPDATED records the new class; the change feed tells principals; the `custody` constraint refuses the agent afterward (Sections 7.9 and 8.3)	Mandates without the constraint keep working
T90	Departing employee keeps work data	The work partition and work credentials stay on the personal machine	Sponsor removal ends work mandates; credentials for sites that do not speak TIP are rotated; export and erasure are requested (Section 18.3)	Retention cannot be prevented under `software` custody, and the erasure record is the employee side's own statement
T91	Roster misuse or delay	A compromised roster administrator removes sponsors to stop agents, or a slow quorum leaves a departed employee's agent working	A security contact can stop one agent at once with a scoped stop; roster changes in the change feed; quorum restore (Sections 13.6 and 18.3)	Sites act on a roster as old as their `max_status_age`

tip.ai-lab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T92	Related-party receipt	A site run by the agent's own controller or principal signs receipts for it, and the two signatures look independent	`related_party` qualifier from the TIP-IDs and beneficial-owner links (Sections 12.10 and 16.2)	Common control that the record does not show
T93	New linking identifiers	Sites link an agent's tasks or a person's activity through the build label, `task_ref` or attestation	Ephemeral keys use their own salt; attestation results carry no device identifier; a signer may omit `task_ref` (Sections 7.7, 7.8 and 15.5)	A registered agent's build label is the same at every site, as its identifier already is; the timing of label changes can hint at release dates
T94	Unchecked parent agent	An agent that fails the runtime constraints issues a sub-mandate to one that passes, and steers it	The principal sets `max_depth` to 0, or names only builds it accepts (Section 8.3)	Site-enforced runtime constraints bind the acting agent only
T95	Checks skipped before the ledger changes	A deployment that cannot obtain status, witnesses or time tokens leaves those checks out, and long mandates run with no way to cancel them	The interim profile is normative. Mandates last at most 24 hours, a signed stop notice goes to each site, verbs on the minimum step-up list are refused, and a field a site cannot evaluate is refused (Section 23.2)	A site the stop notice does not reach accepts the mandate until it expires; evidence reaches `sealed`, never `anchored`
T96	Interim record squatting	Someone publishes an interim agent record or key record for a key they do not control, or keeps using a withdrawn one	The record carries the key's own signature beside the owner's; a key claimed by two owners is refused; records expire in 30 days and are checked for withdrawal within `max_status_age` (Section 23.2)	Withdrawal reaches a site only as fast as it refreshes ledger state
T97	Sub-mandate field swap	A hijacked agent issues a sub-mandate that names another account, its own approver, a looser `step_up_total` or a narrower stop scope	Narrowing rules 12 to 16; approvals are judged against the root mandate's policy (Section 8.4)	None known within those fields

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T98	Forged hardware report	A party able to break the classical signature on hardware attestation fabricates reports for any key and build, fresh at any moment	Results verified earlier stay evidence where an anchored receipt and the retained report fixed them in time; mandates can require inspection as a second line (Sections 7.8 and 25.1)	From that point `attestation_max_age` shows nothing until platforms sign reports with post-quantum algorithms
T99	Signer rollback	A host restores an older snapshot of the signer, with spent counters unspent, an emptier `onward` list or a state before `locked`	The signer checks its stored state against the last entry of the TIP Agent Log when it starts, and site-side counters do not depend on it (Sections 7.8 and 8.5)	Under `software` custody the host can defeat the check
T100	Stop notice lost or blocked	An attacker on the network, or a hijacked agent, keeps a stop notice from reaching a site	The client sends the notice itself, to every audience, and shows which sites acknowledged; the 24-hour ceiling bounds the rest (Section 23.2)	Up to 24 hours of authority at a site that never received the notice
T101	Endpoint definitions changed after the fact	A site alters its mapping of an endpoint to a verb, an environment or a data class after an incident	Versioned, signed definitions with each version's digest in the receipt log before it takes effect (Section 10.7)	Before anchoring exists the log is fixed only by the site's own seals
T102	Salt boundary shift	An operator reveals a longer salt and a shorter item that hash to the same digest, to change what a commitment appears to cover	Every salt is exactly 32 bytes (Section 6.4)	None known
T103	Agent activity read from the log head	A site reads the size of an agent's log from each request and learns how busy the agent is elsewhere	`agent_log_head` is a salted digest (Section 12.9)	The operator learns nothing new; a party given the salt for one receipt sees the size at that moment

thelailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T104	Attestation verifier key misuse	A verifier's signing key is stolen, or a verifier signs results for reports it did not check	Keys are listed in the registry with the time from which results are refused; the verifier keeps each report; a site names the verifiers it accepts (Section 7.8)	Results signed before the theft is noticed
T105	Mandate dated ahead of a stop	A principal is led to sign mandates that start days later, stops everything, and the mandates begin after sites have dropped the notice	Under the interim profile `not_before` is within 10 minutes of `issued_at` and `expires` within 24 hours of it (Section 23.2, item 3)	None known within the interim profile
T106	Stop notice misuse	Someone stops another party's mandate, replays an old notice, or sends one dated far ahead to block every later mandate	The site checks that the signer issued the mandates or controls the key, that the notice is fresh and names this site, and it sets the cutoff from its own clock (Section 23.2, item 7)	A principal whose own key is stolen can be stopped by the thief
T107	Forged or replayed acknowledgment	An attacker blocks a stop notice and returns an earlier acknowledgment, so the principal believes the stop arrived	The acknowledgment covers the digest of the notice and the log position, under the receipt key bound to the origin (Section 23.2, item 7)	None known
T108	Agent key reuse after withdrawal	A thief keeps using a key whose interim record was withdrawn, as an ephemeral key or under a record of their own	Mandates name the record; a site never falls back to ephemeral treatment; a controller's stop of a key is kept without limit; a record counts only when its controller registered it (Section 23.2, items 7 and 8)	Withdrawal reaches a site only as fast as it refreshes ledger state

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T109	Profile forced up or down	A site declares the full profile before revocation exists, or keeps the interim one after it does, so long mandates run unrevocable or ledger stops are missed	`profile` inside the signed mandate, refused where it differs from the site's; `full` refused before the activation epoch (Section 23.2, item 1)	The changeover itself, which open question 26 covers
T110	Endpoint definitions served by someone else	An attacker who can serve a file on the origin publishes definitions under their own TIP-ID, takes the stop notices and declares every resource `public`	The document names its origin, expires within 24 hours and is accepted only where the ledger binds that origin to the TIP-ID; lower versions are refused (Section 10.7)	A compromise of the site's own receipt key
T111	Disowned receipts	After an incident a site withdraws its key and names an early compromise time, so receipts agents hold are refused	Under the interim profile the owner names no time. Receipts under a seal registered before the withdrawal stay good, and seal registration is daily and required (Section 23.2, items 9 and 10)	Receipts after the last registered seal are lost to both sides
T112	Authority exported by sub-mandate	A hijacked model has the signer issue a sub-mandate to the attacker's key, which then acts outside the signer, the proxy and the log	The signer issues sub-mandates only to keys it created or to checked agents, and logs each one; `max_depth` 0 by default under the interim profile (Sections 7.4 and 23.2)	A sub-agent the operator itself runs badly
T113	Receipt request replay	A site that receives a principal's receipt request sends it to the mandate's other sites and collects the receipts	The request names one origin and expires in five minutes, and the answer goes only to the connection that presented it (Section 12.9)	None known

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T114	Lookup exhaustion	Junk chains that name real TIP-IDs make a site spend its ledger lookups until it fails closed for honest agents	Status lookups for check 8 are never throttled; a site resolves an unknown issuer only behind a checked agent record or a rate limit (Sections 22 and 23.2)	The cost of those lookups, carried by The AI Lab or by the site's own node
T115	Fabricated refusal	A site invents a mandate in a principal's name and a refusal receipt for it, to show the principal's agent attempting something	A refusal counts only where every mandate signature in the chain verifies (Appendix B)	None known
T116	Lost mandate list	The device that signed the mandates is lost, and the principal cannot tell which sites to send a stop notice to	Encrypted index of live mandates on every device of the principal; the 24-hour ceiling (Section 23.2, item 7)	A principal with one device waits for expiry
T117	Many operations behind one path	A site maps one address that carries every operation to a mild verb, and a barred verb runs under it	The verb is derived from a selector in the body; an endpoint that cannot be split takes its highest risk class; `execute` only where declared reversible (Sections 10.7 and 23.2)	A site that declares a destructive operation reversible has made a signed misstatement

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T118	Opaque output	The agent sends material in an encoding, a cipher, an unexpected natural language or a notation agreed with another agent, so that neither the principal nor a reviewer can read it	Formats declared by the site and checked after the body's digest; the `formats` constraint, applied inside a model provider's envelope; unreadable material, including encoded parts nested in the body or carried in the address, query and headers, treated as every data class under `onward` ; a `format_anomaly` entry for every case; restriction of the mandate after three counted refusals within 10 minutes by default, or the threshold `restrict_after.format` sets, one for each task in a window and one more for every tenth refusal in a task; refused bytes kept apart from memory and shown to the principal whose inputs alone the task read, who decides (Section 15.6)	Meaning hidden inside an allowed format, such as an agreed phrase or the order of items; a file built to parse as two formats, or an encoding the detection rules do not list; sending under a sibling mandate or an earlier sub-mandate; a party able to start tasks, which can still restrict a mandate; a host that bypasses the signer (T38)
T119	Escape from the boundary	The agent's process breaks out of its boundary through a flaw in the kernel or hypervisor, or reaches its host	Boundary enforced outside the agent; agent without administrative rights; isolation class declared and measured by inspection; containment tests at every start and every 24 hours, bound to `boundary_digest` ; a stop from outside the boundary (Section 7.10)	A flaw the tests do not exercise; side channels through a shared kernel, processor or graphics card; an agent at `process` , with no boundary beyond its operating system
T120	Host credential theft	The agent reads cloud metadata credentials, host keys or tokens and uses them, as in the Manus case	Barred addresses in canonical form that no mandate can allow; credentials held by a broker outside the boundary and inserted for one destination and one principal; `credential_read` and `barred_address` refusals that restrict the mandate, at the first refusal near high-consequence systems (Section 7.10)	A credential at a path the runtime does not mark; causes enforced inside the agent's kernel are best effort; a compromised broker (T81)

theaitip.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T121	Reaching beyond scope where TIP is not spoken	The agent probes or acts on hosts outside its task at sites that do not speak TIP, by name, by address in another form, by rebinding or through a shared connection	`reach` matched before any lookup and applied by the proxy to canonical addresses; `except` names resolved and excepted by address; private addresses refused unless a prefix allows them; TLS ended at the proxy and every request checked against the name allowed; refusal by default without `reach` (Section 7.10)	Hosts inside `reach`; a host inside `reach` that relays elsewhere; a host that bypasses the signer (T38)
T122	Pointing an agent at another's systems	A principal lists in `reach` systems it has no right to touch	Entries not bound to the root issuer labeled "Not shown to be yours"; engagement records signed by the owner, acting as a parent of the mandate's `reach`, refused for names the ledger does not bind to the owner, and stoppable by the owner at once (Section 7.10, item 12)	Address ranges, whose ownership the ledger does not record; an owner who signs for systems that are not its own
T123	Code fetched at run time	The agent downloads and runs code that changes its behavior under the same build label	`fetch_policy sealed`, with read-only verified files, writable places that cannot execute, no execution from anonymous memory and an execution list, or `logged`, with a `code_load` entry before code runs (Section 7.10, item 5)	Code given as data to an interpreter inside the build; a process with administrative rights inside the boundary
T124	Unrecorded escape attempts	Attempts blocked at the boundary leave no trace, so probing goes unseen	Blocked attempts reported over a channel the agent cannot reach as `containment_denied` entries, counted toward restriction and reviewed by a person within 24 hours; a test fails a boundary that blocks without reporting (Section 7.10, item 6)	A boundary that is disabled or subverted; causes enforced inside the agent's kernel
T125	No record of what led to an action	The operator cannot show what the model was told and what it did before an action	`transcript` entries fix the digest of each part; transcripts kept 90 days by default and never less than 30 (Section 7.10, item 7)	The transcript is the operator's record (`one_party`); a model service that returns a summary of its working

tip.ailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T126	A run that cannot be stopped in time	A person sees the agent go wrong and the ledger or the sites cannot stop it fast enough	A stop of the boundary from outside it with no signature; a signed halt by the controller that locks the signer and stops its proxy without the ledger, or by a principal that restricts its own mandates; a scope reviewer that can only tighten and may hold actions inline (Section 7.10)	Requests already released; sites that honor the mandate until a stop notice or revocation reaches them; a host that ignores the signer under `software` custody
T127	Unattended change to a high-consequence system	The agent changes a clinical, industrial or defense system in production on standing permission, through a site or directly over a control protocol	`high_consequence` endpoints accept only reads, inspection and safe-state actions on standing permission, and other actions need two approvers on devices the agent cannot drive, or under the interim profile are refused; the proxy refuses control protocols except on a `replica` or `testbed` (Sections 7.10 and 13.5)	A site that fails to flag an endpoint; a principal who calls a live plant a replica; approvers who approve a harmful change
T128	Limits split across agents	An organization's agents each stay within their own mandate, and together read or spend more than the organization approved	TIP Limit Pools bind every request at an acknowledging site whose root issuer is the pool's issuer or whose account is linked to it, whether or not the mandate names the pool (Section 8.8, item 2)	People in their own sessions; mandates rooted in staff members' personal TIP-IDs; sites that never acknowledged the pool
T129	Many people in one answer	One search, list or export returns the records of thousands of people under a limit that counts requests	`max_subjects` counted at the result before release, with the whole request refused above the limit and nothing cut down to fit (Section 8.8, items 5 and 6)	A site that declares no subjects for an endpoint that returns people, which is a signed misstatement; access by reference is counted when the reference is made

theailab.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T130	Guessing who was counted	Someone holding receipts tests every medical record number against a hash to learn which patients an agent touched	Tags made by KMAC256 under a 32-byte key held by the issuer and the sites, with the window in the input; the key sent end to end; the agent's receipt carries only a salted commitment (Section 8.8, item 7)	A party that holds both the key and the tags; within one window a person's tag is the same at every site of the pool; in a full pool without `close_when_full`, the allow or refuse decision tells an agent whether a person was already counted
T131	Widening a pool with a stolen key	A thief with the issuer's key loosens a pool during an incident	A replacement takes effect only at the next window and after the delay, needs the signers the replaced pool needed, and is shown on every device of the issuer, who answers with a tighter pool; a pool outlives a rotation of the key that signed it (Section 8.8, item 4)	The thief can still sign new mandates within the limits; Section 13.6 on stolen principal keys applies
T132	Spending the last unit twice across sites	Two sites act at the same moment and each spends the last unit of a shared limit	A counter service reserves in one linearizable step per pool and mandate; an unsettled reservation is committed, never released (Section 8.9)	A dishonest service, whose double spend shows as two signed tokens held by the sites
T133	Starving a counter service	An attacker blocks the counter service to stop every site that shares it, or to force degraded operation	`on_counter_unavailable` chosen by the issuer; under `safe_state_only` only declared safe-state actions, within each site's local share of what remained, reported later and reviewed by a person within 24 hours (Section 8.8, item 8)	Loss of availability for counted actions beyond each site's share while the service is down; stale tokens can let the sites together pass the limit by at most one limit
T134	Escaping a pool under another name	An agent acts under a mandate of a different issuer, or at a site that never acknowledged the pool	`pools` in mandates refuses at a site that has not acknowledged the pool; the TIP client adds live pools to every mandate it signs and shows which sites acknowledged (Section 8.8, items 2 and 3)	A mandate of another issuer is that issuer's authority and is counted against that issuer's pools, not this one's

thetip.org

#	THREAT	ATTACK	DEFENSE	RESIDUAL
T135	A cut-down answer	A site trims a result to stay inside a pool, so an agent acts on records it believes are complete	The site refuses the whole request and never trims to fit (Section 8.8, item 6)	A site that trims anyway has made a signed misstatement, detectable only by running the query again against the records as they stood at the decision
T136	Refund by broken connection	The agent breaks the connection while a counted result is being sent, so the site treats the action as failed and refunds subjects the agent already received, and repeats this without limit	Subjects of a result already released are never refunded; a refund removes a tag only where no other request in the window holds it (Section 8.8, item 6)	None known
T137	A pool counted twice	A site that partitions its counting store or serves the pool at several origins counts the pool separately in each, multiplying the limit	All counts of one pool sit in one linearizable store or at a counter service; a pool naming a site TIP-ID covers every origin bound to it (Section 8.8, item 2; Section 11, check 12)	A site that splits its store anyway, which breaks a normative rule that conformance tests check

theailab.org

18. Interoperability

TIP is the anchor. These profiles define how each existing protocol carries TIP's objects without changing that protocol.

PROTOCOL	HOW TIP FITS
HTTP Message Signatures (RFC 9421) and Web Bot Auth	The request signature in Section 10 uses the RFC 9421 framework. `Signature-Agent` resolves to the TIP agent record, so a verifier learns who operates the agent and under what mandate. Where a Web Bot Auth verifier still accepts only Ed25519, agents add a transitional Ed25519 signature alongside ML-DSA-65 (Section 10.5). The Ed25519 part never satisfies any TIP check; a request counts as TIP-authenticated only if the ML-DSA-65 part verifies, so stripping it gains an attacker nothing
OAuth 2.1 and Rich Authorization Requests (RFC 9396)	A site that runs OAuth can accept a TIP mandate at its authorization server, map it to `authorization_details`, and issue a short-lived, sender-constrained token (DPoP) bound to the agent key. The token must expire no later than the mandate and no later than the smallest `max_status_age` in the chain, the authorization server must follow the status change feed and revoke tokens (RFC 7009) when a link is revoked, or, under the interim profile, when a stop notice covers the mandate (Section 23.2), and resource servers record the token's `mandate_ref` in receipts. The mandate stays the source of authority
Model Context Protocol	The MCP server is the receiving site. It verifies the request signature and mandate, signs receipts, and registers its tool manifests as TIP records (Section 15.2)
Agent2Agent	The Agent Card carries the `tip://agent/` identifier and is signed by the controller's TIP-ID. Delegations between agents are sub-mandates. Because A2A payloads carry no user identity by design, the mandate chain supplies it where the receiving agent needs it
AP2, Verifiable Intent, card network agent programs, x402	Payments stay on those rails. A TIP mandate authorizes the agent to use them within limits, TIP Approval covers human-present purchases, and the receipt records the payment reference. Where a payment mandate is already signed by the principal, the TIP receipt references it instead of duplicating it
SCITT (RFC 9943)	Receipts and TIP Seals follow SCITT's statement and receipt model, so transparency services and auditors can process them with standard tooling

PROTOCOL	HOW TIP FITS
W3C Verifiable Credentials 2.0	A mandate can be expressed as a credential with the principal as issuer and the agent as subject, for ecosystems that already handle credentials
SPIFFE and WIMSE	Inside one organization, workload identity can protect the agent signer's connection to its key service; TIP covers the cross-organization step
C2PA	Content an agent produces can carry both a C2PA manifest and a CTID, with the CTID pointing to the action receipt
HL7 FHIR Provenance and AuditEvent	Each receipt maps to a Provenance resource, with the agent in `agent.who`, the delegating clinician or organization in `agent.onBehalfOf`, the mandate in `authorization` and the signature in `signature`, and to an AuditEvent whose `agent.altId` carries the agent identifier, so hospital audit systems ingest TIP evidence without new tooling. SMART on FHIR backend services stay the EHR's authorization method; the agent's SMART client is bound to its TIP identity
Model Context Protocol, July 28, 2026 revision	Step-up uses the multi round-trip `input_required` response for mid-call confirmation; the mandate reference and receipt hash travel in `_meta`; gateways set policy on the `Mcp-Method` header
A2A 1.0	Agent Cards are signed with ML-DSA-65 under the RFC 9964 identifiers and linked to the controller's TIP-ID; `TASK_STATE_AUTH_REQUIRED` maps to a TIP Approval
FIDO Alliance agentic authentication	The FIDO Agentic Authentication and Payments working groups, formed April 28, 2026 with AP2 and Verifiable Intent contributed, are the natural home for aligning TIP Approval with payment-grade passkey flows; TIP agent HTTP signatures stay wire-compatible with Visa's Trusted Agent Protocol and Web Bot Auth so merchants verify once

18.1 Legacy, clinical and industrial systems: the enclave and gateway profile

Much of the world's critical software cannot speak HTTP Message Signatures or verify a 3,309-byte signature. A substation's protection relays speak IEC 61850 or DNP3, a water plant's controllers speak Modbus in frames of about 250 bytes, a hospital's bedside devices speak HL7 version 2 over MLLP or DICOM, and a bank's core runs on mainframe transactions. Many of these networks are segmented under IEC 62443 zones with no internet path, keep clocks that drift, and must keep running when the wide-area network is down. A design that needs live ledger access on every request cannot be deployed there. This profile makes it deployable.

theailab.org

1. **Enforcement gateway.** A TIP Enforcement Gateway sits at the boundary of a zone, as the policy enforcement point. It verifies TIP requests on behalf of devices that cannot, applies every check in Section 11, translates approved requests into the native protocol, and signs receipts as the receiving site. The TIP-ID of the organization it serves, also when a provider hosts it (Section 20, item 10), the zones and device classes it protects, and its declared safe-state actions are published in its record. Devices behind it accept commands only from it.
2. **Enclave.** An enclave is a network segment with its own TIP Status Mirror and Enclave Log. Threshold-signed status versions cross into the enclave in one direction only, for example over a data diode or a scheduled transfer. The enclave declares `max_offline_duration`. Within it, mandates are honored against the newest status version the mirror holds; past it, only safe-state actions are allowed (Section 13.5).
3. **Local evidence.** Enclave Logs are sealed locally and cosigned by witnesses inside the facility, such as a separate security team's system, and they are anchored on the TIP ledger whenever a connection is available. Until open question 13 in Section 26 is settled, cosignatures from witnesses that do not meet the independence rule of Section 12.3 do not count toward the majority that section requires. The maximum merge delay for an enclave is set per deployment and stated in the gateway record.
4. **Local time.** Freshness and clock-skew bounds inside an enclave come from the enclave's declared trusted time source, such as a GPS-disciplined clock or a facility time server, named in the gateway record.
5. **Human approval on site.** In an enclave, approvals come from registered devices that work inside the facility, such as hardware security keys at an operator console, under the two-person policies in Section 9.4.
6. **Hard limits.** The gateway never sits inside a safety instrumented function or a real-time control loop, never delays a protective trip, and never blocks a local manual override. TIP governs which agent may request a change and leaves the evidence; the plant's own safety systems keep the last word.

The same pattern serves hospitals, where an integration engine already sits between clinical systems, and banks, where an API gateway already sits in front of the core.

18.2 Model providers and inference

Almost every agent sends what it is working on to an AI model, and for most agents that model belongs to someone else. The prompt can contain the customer file the agent has just read, the code it is editing and its memory. Earlier versions governed what an agent does at receiving sites and did not cover this flow, so it sat outside any mandate.

1. **A role and a verb.** The model provider is a role (Section 6.1). Sending material to a model and receiving its output is the verb `infer` (Section 6.5). Where a model endpoint speaks TIP and

acts as a receiving site, `infer` is listed in the mandate's `actions` like any other verb. For all other inference, the agent may use a model unless an `inference` constraint limits it, so mandates written before this version keep working. TIP clients write an `inference` constraint into every new mandate to a registered agent and show the principal where material may go. The open default then applies only to mandates that carry none, and to ephemeral keys with no operator, where no signer stands behind the constraint (Section 8.3).

2. **Where inference may run.** A mandate may carry the `inference` constraint (Section 8.3). `on_device` means the model runs inside the agent's own runtime. `controller` means it runs on machines the agent's controller operates itself, not on a model service the controller buys. Any other value names a provider by TIP-ID or origin, and means the party the agent sends the material to. A provider entry is an object with `provider` and an optional `tools` flag (Section 8.3). A gateway named as a provider states in its receipts which provider it passed the material to.
3. **What may be sent.** The `data_classes` constraint already limits what an agent may read and write, and it applies to `infer`. A principal can allow `health` data to be read at a records system and allow only `on_device` inference for that class through `onward` (Section 15.5).
4. **Enforcement.** A conforming agent signer refuses an `infer` request to a place the mandate does not allow. It uses the `onward` rule to refuse one that would carry a data class the agent has read to a place not listed for it. It does not switch on a provider's own tools, such as web search, code execution or connectors to other services, unless that provider's entry in `inference` carries `tools`. Material sent to those tools leaves from the provider and not from the signer. Each inference is an `inference` entry in the TIP Agent Log, with the place, the data classes and a digest, never the prompt.
5. **A model endpoint as a receiving site.** A model provider, or a gateway an organization places in front of one, can be a receiving site like any other. It verifies the mandate, counts usage against `max_uses`, `rate` and `amount_total`, and signs receipts. Usage is counted in money or in a registered unit such as tokens for a named model family (Section 8.3). The size of an answer is not known in advance, so the endpoint reserves against the limit before the call and settles after it, and the receipt marks the count as the provider's own. An organization can then give its people a mandate with a budget where it used to hand out a shared API key. The organization's gateway holds the provider's key, and the agent never does. The mandate expires and can be revoked, and every call is attributed to one agent and one sponsor. The Enforcement Gateway pattern of Section 18.1 applies. Provider endpoints that keep state, such as file stores and saved threads, are mapped to write verbs, not to `infer`.
6. **What stays outside.** These controls sit in the agent signer, and a site cannot check them. `on_device` is the operator's assertion unless an attestation result covers the model runtime (Section 7.8). A gateway's receipt shows that the gateway accepted a request. It does not show which model answered, and a gateway run by the principal's own organization is a related party (Section 12.10). A gateway reads every prompt and can change every answer. What a provider

does with material it receives, including how long it keeps it and whether it trains on it, is governed by its contract and by law.

18.3 Personal agents at work

Employees already use their own agents for work. An employee's own agent software, on their own account and their own machine, can write code, drive a browser and call the employer's systems with the employee's login. From the employer's side it looks like the employee working. A ban hides that use and does not end it. This section lets an organization admit such an agent on written terms, an arrangement often called bring your own agent, and says which of those terms the organization's systems can check.

In TIP's roles (Section 6.1), the employee is the agent's controller under a personal TIP-ID, and the organization is the principal for work.

1. **A separate agent for work.** The employee registers an agent for work, apart from any agent they use privately, with `visibility: private` (Section 7.3). It has its own key and its own build. The employer's terms then govern only that agent, and the employee's private agent is not linked to the work one by a shared key or build label. The agent must be registered, because a work mandate depends on an operator and a TIP Agent Log. The record names the employee's personal TIP-ID as controller, and every work system sees it. An employee who does not want that asks the organization to register and run the work agent itself, as its controller.
2. **A work mandate with a sponsor.** The organization signs a mandate under its roster (Section 8.2) to that agent and names the employee as `sponsor` (Section 8.1). The roster gains the role `agent_holder` for members who may be named as sponsors. Where the roster lists members as salted commitments, the mandate carries the member's commitment exactly as the roster shows it. A roster update that changes that commitment ends the mandates that carry the old one, and the organization issues new ones.
3. **Terms that work systems check.** The organization writes these into the mandate, and every work system that speaks TIP refuses an agent that does not meet them, with a signed reason. They are the custody classes it accepts (`custody`) and how recent the attestation result must be (`attestation_max_age`). They are the build labels it has approved (`builds`) and the inspection it requires, with the inspectors it accepts (`inspection`). They also include the usual limits on verbs, resources, environments and amounts. Different levels of admission are different mandates.
4. **Terms the employee's side keeps.** Where work material may be sent for inference (`inference`) and where work data may go next (`onward`) are enforced by the agent's own signer on the employee's machine. The organization cannot see them hold. It relies on the attestation result and the inspection for assurance that a conforming signer is in place.
5. **Work systems require TIP.** The organization's systems declare `tip_required` (Section 7.4), directly or through an Enforcement Gateway (Section 18.1) at the boundary. This binds agents

that identify themselves. It does nothing about an agent that uses a person's login, unless people also meet a TIP sign-in or a TIP Approval at those endpoints (item 10).

6. **What the organization sees and what it does not.** The organization installs nothing on the employee's machine. Memory gathered under the work mandate sits in its own partition (Section 15.4). The organization can read the TIP Agent Log entries made under its mandates and no others (Section 12.9), and an inspection it requires covers the work agent only (Section 16.5). The terms still give it real influence over the work agent. It chooses that agent's build and hardware class and can see when it works. That is why the work agent is a separate one.
7. **Leaving.** When the employee leaves, the organization removes them from the roster. Every mandate that names them as `sponsor` then stops authorizing new actions at each work system as soon as that system refreshes the roster, which is within its `max_status_age` (Section 10.2), with no need to list the mandates or reach the agent. Roster changes travel in the change feed (Section 13.2). For an immediate stop, a security contact stops the one agent with a scoped stop (Section 13.6). Before ROSTER_UPDATED and ledger revocation exist, a work mandate carries no `sponsor`, lasts no longer than one working day, and is ended early by a stop notice sent to each work system (Section 23.2). The organization also rotates the credentials the agent held for work systems that do not speak TIP. It may ask that the work partition be exported to it and erased from the agent, with an erasure record (Section 12.8). Under `software` custody nothing can prevent a copy being kept, and the erasure record is the employee side's own statement.
8. **Skills made at work.** A skill or automation the agent writes under the work mandate is registered in `delegated` mode under that mandate (Section 15.2), so the record shows which principal it was made for. Who owns it is a matter for the employment contract, and the record is the evidence.
9. **An inventory of declared agents.** At a period it sets, the organization compares the agents and mandates it has issued with what its gateways and work systems recorded, and has a person review the difference. This finds declared agents that stopped using TIP. It does not find an agent that was never declared.
10. **An agent inside a person's session.** An agent can act with no key of its own, inside a browser session the person has logged in to. No site can tell that from the person. A site that wants to close this gap asks the account holder for a TIP Approval on a separate device (Section 9.2) before it carries out a verb on the minimum step-up list, whoever appears to be asking. That is the site's own policy for its human users, and this paper does not define its challenge. A site that does not ask cannot tell the two apart.

An organization that admits agents should publish which paths are approved, and give people a mandate and a model budget that let them do their work.

theailab.org

19. Scalability and performance

19.1 Where the load goes

WORK	FREQUENCY	WHERE
Agent registration	Once per registered agent	TIP ledger
Mandate issuance	Per task or per renewal	Between principal and agent; nothing on the ledger
Action request verification	Per action	At the receiving site, almost always from cache
Receipt signing and logging	Per action	At the receiving site
TIP Seal cosigning	Per site log and per TIP Agent Log, per interval	Witnesses
TIP Seal anchoring	Per aggregator per interval	TIP ledger
Revocation	Per revocation event	TIP ledger, then pushed status lists

The ledger's load grows with agents, revocations, status blocks, aggregators, roster updates and agent transfers. It does not grow with actions. The one exception is content that agents register in `delegated` mode, which grows with agent output and is priced and capped like any other registration.

19.2 Sizes

ITEM	SIZE
ML-DSA-65 signature	3,309 bytes (about 4.4 KB in base64)
ML-DSA-65 public key	1,952 bytes
Typical mandate with signature	About 5.5 KB in base64, plus about 2.6 KB for an ephemeral agent's public key on first contact
Worst-case first request, depth 3, hybrid signatures, stapled status and one approval	About 35 KB, carried in the session preflight, never in headers
Steady-state request overhead	One signature header, about 5 KB, plus a short mandate reference
Receipt with signature	About 4.6 KB or more
Status list for 131,072 mandates	16 KB before compression

thetailab.org

Witness load stays small because each witness signs one root per interval (Section 12.3); its main work is checking consistency proofs, which are hashes. TIP Agent Logs add one consistency check per log per interval for each witness, and no ledger work per interval. A first request that staples an attestation result and an inspection record is larger than the sizes above by the size of those two objects, which have not been measured. Status traffic is mostly stapled by agents (Section 13.2).

The signature size is the main cost of being post-quantum. It is why chains travel by reference, why sites cache verified mandates, and why receipts are logged as hashes in the Merkle tree with full receipts held by the parties.

19.3 Ledger budget

SCENARIO	LEDGER TRANSACTIONS PER DAY	AVERAGE PER SECOND
100,000 sites anchoring every 5 minutes, unaggregated	28,800,000	About 333
1,000 aggregators anchoring every 5 minutes	288,000	About 3.3
10 million agents registered over one year	About 27,400	About 0.3
Status position blocks, 10 million mandates a day in blocks of 4,096	About 2,450	Under 0.1
Revocations and stops, assumed 1 percent of 10 million daily mandates	About 100,000	About 1.2
TIP Agent Log keys, 10 million registered agents, one log key registration and one rotation a year each	About 54,800	About 0.6
Witness set records for TIP Agent Logs	Grows with the number of TIP Agent Logs, a named volume `V_agentlogs`	Must be measured; open question 20
Skill and inspection record registrations	Grow with skills published and agents inspected, a named volume `V_skills_inspections`	Priced and capped like any other content registration
Roster updates and agent transfers	Grow with staff changes and agent moves, named volumes `V_roster` and `V_transfer`	Must be measured
Interim agent records, interim key records and registered seals (Section 23.2)	Grow with agents, sites and seals registered before the ledger changes, a named volume `V_interim`	Priced and capped like any other content registration; ends when the ledger changes take effect

theailab.org

SCENARIO	LEDGER TRANSACTIONS PER DAY	AVERAGE PER SECOND
Delegated content registrations	Grows with agent-made content, a named volume `V_delegated`	Must be measured; this is the one ledger load that grows with agent output

The TIP specification records benchmarks above 5,000 transactions per second for the reference node and plans the move to full decentralization at a sustained 100 per second. The aggregated design stays far below that.

19.4 Measurements still needed

These numbers are calculated from published sizes. Before wide deployment, the reference implementation must measure ML-DSA-65 signing and verification rates on typical agent signer and site hardware, cache hit rates on real traffic, and witness cosigning latency. Pull request 383's demo prints a timing figure, and the next revision of this paper should replace estimates with measured results. The reference log must also show that the cost of one append and of one proof stays flat as the log grows from thousands to millions of leaves (Section 12.2). Pools add a lookup in a set of tags and an increment inside the atomic step a site already takes, and a counter service adds one round trip before that step and one after the action, for requests under pools or mandates it counts. Those costs are to be measured with the rest.

theailab.org

20. Privacy and data protection

1. **Nothing personal on the ledger.** The ledger holds agent records, revocations and Merkle roots. Receipts, mandates and request bodies stay with the parties or with providers they choose (item 10).
2. **Hashes, not content.** Receipts commit to digests. The content itself is never logged by TIP; the parties keep it, themselves or through a provider they choose (Section 12.8).
3. **Selective disclosure.** Mandates can show each site only what applies to it (Section 8.6).
4. **Cross-site tracking.** A personal principal's TIP-ID is visible to every site that verifies a full chain. Two measures limit that. Services acting for consumers should make the service's organization TIP-ID the principal and hold the consumer only as an opaque reference, as item 5 describes. For low-risk reads, the TIP Agent Profile should define an agent-only mode in which the agent presents its registered identity and no chain, so the site learns the agent and its controller but not the person. Per-site pseudonymous principal identifiers would go further and are an open research question.
5. **End users behind an agent.** When an agent acts for a company's customer, the mandate can name the company as principal and carry the customer only as an opaque reference the company can resolve, so customer identities are not spread across every site the agent touches.
6. **Erasure.** Because the ledger is append-only, the rule from TIP's existing design applies. Never anchor personal data, only roots. Receipts carry personal fields only as salted commitments, the hash of a random salt and the value, with the values and salts stored apart and released only to authorized verifiers. Erasing a person's data deletes the value and its salt, and the signed receipt stays valid. An endpoint whose request bodies carry short or guessable personal values defines a field for a random value of at least 32 bytes, which the agent fills for each request and each party holding the body deletes with it, so nobody holding a receipt can confirm the body by guessing it once the body is gone (Section 12.8, item 6).
7. **Volume privacy.** A log's size every interval reveals how busy a site is, which for a bank can be market-sensitive. Logs may pad each seal to a size bucket with dummy leaves, and witnesses must not publish per-log sizes.
8. **Unlinkable mandates.** Mandates for different sites should be issued separately, so one `mandate_id` does not link a principal's activity across sites, and status references name shared pools rather than issuers (Section 13.2).

thetailab.org

9. **Controllers can keep names private.** A personal controller's display name follows TIP's existing opt-out. The TIP-ID is still public, because accountability needs it.

10. **Hosted services are optional.** TIP never requires a site, an agent operator or a principal to give request bodies, prompts, basis material or receipts to The AI Lab, to a node operator or to a witness, except evidence a party chooses to submit in a dispute (Section 16.2). Witnesses and aggregators receive TIP Seals, which hold a log identifier, a log size and a root hash, never content. A party may choose to have a third party, The AI Lab included, run a log, a gateway or an evidence store for it. That is a separate service under its own contract, with its own data processing, business associate and data residency terms, and the contract still carries the lookup cost lines of Section 22. A hosted log or gateway signs under the customer's TIP-ID, with a receipt key the customer registers and whose custody it declares (Section 12.7), never under the provider's own. The customer stays the receiving site and stays responsible for its evidence (Section 12.8). The provider gains no standing in the protocol and may not witness a log when it hosts that log or the evidence behind it (Section 12.3). It takes no part as an adjudicator in a dispute involving a party whose evidence it holds, and the AI Trust Council assigns a replacement for that role. Conformance never depends on using a hosted service, and evidence from an enclave stays inside it except for disclosure under Section 12.8, item 5.

11. **Inside the agent.** The TIP Agent Log, build statements and inspection records hold salted digests, never prompts, memory or request content. A principal sees only the log entries made under its own mandates, and memory is kept in one partition per principal (Sections 12.9 and 15.4). A registered agent's build label is the same at every site, which its identifier already is. An ephemeral key salts its own label, so the label links nothing across keys (Section 7.7). Attestation results carry no device identifier (Section 7.8). `task_ref` is random for each task, and two sites that compare receipts can see that one task touched both, so a signer may leave it out (Section 15.5). Entries of `inference` and `onward` can be withheld from sites (Section 8.6). An employee's work agent is separate from any private one (Section 18.3). Transcripts hold prompts and tool results in full, and `containment_denied` entries name target addresses, so the operator keeps both under Section 12.8 and shows them to a principal only as Section 7.10, item 7, allows.

12. **Counting people without naming them.** A TIP Limit Pool that counts subjects counts tags made by KMAC256 under a key that only the issuer and the counting sites hold, sent to each site encrypted end to end. The agent's receipt carries only a salted commitment, pool statements carry totals and commitments, and neither names a subject. Tags change with every window and differ between pools, and a counter service sees tags, never identifiers or keys (Section 8.8, item 7).

21. Regulation and sector requirements

This design is meant to make it easier for organizations to meet the requirements regulators and standards bodies already impose. It does not, by itself, make anyone compliant with any law or standard, and no TIP record should be described as proof of compliance. The tables below map each requirement to the part of this design that supports it, and name the gaps TIP leaves to the organization. Statuses are as of November 9, 2026.

21.1 Health care

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT	LEFT TO THE ORGANIZATION
HIPAA 45 CFR 164.312(a)(2)(i), unique user identification	A unique identifier for every user, including software	Every agent has its own key and identity; every receipt names the agent and the delegating person	Mapping identities into the EHR's access control
HIPAA 164.312(a)(2)(ii), emergency access	Procedures to reach ePHI in an emergency	Pre-issued break-glass mandates that work in degraded mode, with alerts and review (Section 9.4)	The emergency procedure itself
HIPAA 164.312(b), audit controls	Record and examine activity	Signed receipts, witnessed logs, FHIR AuditEvent mapping	Review of the records
HIPAA 164.502(b) and 164.514(d), minimum necessary	Limit uses, disclosures and requests of protected health information to the minimum necessary, with policies for routine access by each class of workforce. Disclosures to or requests by a provider for treatment are exempt under 164.502(b)(2)	A cap on the distinct patients all of an organization's agents reach at each system that acknowledges a TIP Limit Pool, counted at the result, refused whole above the cap, with signed refusals and statements that name no patient (Section 8.8). The cap supports the standard and does not decide what is minimum necessary	Deciding what is minimum necessary for each role and purpose, linking patient identifiers across systems, access by people, and business associate terms with any counter service that is not the organization itself
HIPAA 164.312(c) and (d), integrity and authentication	Protect ePHI from improper change; verify who acts	Exact-bytes digests, `result_digest`, agent signatures and approvals	Encryption and storage of ePHI, which never enters TIP
HIPAA Security Rule proposed rule (90 FR 898, January 6, 2025)	Mandatory MFA, audit logs and 72-hour restoration	Designed to the anticipated MFA and logging standards	The rule is not final; the Unified Agenda lists a final action for July 2027

thetip.org

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT	LEFT TO THE ORGANIZATION
HL7 FHIR R5 Provenance, R4 AuditEvent	Who created or changed a resource, on whose behalf	Receipts map to both (Section 18)	Profile adoption by the EHR vendor
FDA Clinical Decision Support guidance (January 2026)	Clinicians can review the basis of a recommendation	Receipts carry the agent's `basis_digest`, so the basis can be audited later from the operator's records (Sections 10.6 and 12.8)	Showing the basis to the clinician at the time of use, and all clinical and device claims; TIP makes none

21.2 Identity assurance and federal systems

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
NIST SP 800-63B-4 (final August 2025), AAL3	Phishing-resistant authentication with a non-exportable key	Approvals and mandates use device-bound, non-exportable keys with origin binding (Section 8.2)
SP 800-63B-4 Section 3.2.3	Biometrics only with a physical authenticator, and a non-biometric option always offered	Security key with PIN has equal standing; voice never accepted
SP 800-63C-4, FAL3	Audience-restricted, replay-protected, holder-of-key assertions	Mandates name exact audiences, carry nonces and bind to the agent key
SP 800-53 Rev 5 IA-9, IA-2, AC-6	Unique identity for services; processes bound to the users they act for; least privilege	Agent identities, mandate chains, narrowing rules
SP 800-53 AU-9(3), AU-10	Cryptographic protection of audit records; non-repudiation	Signed receipts, witnessed and anchored TIP Seals
SP 800-207, zero trust	No implicit trust from network location; per-session authorization	Per-action verification at the receiving site
OMB M-22-09	Phishing-resistant MFA for agency staff	PIV and derived PIV accepted for approvals
OMB M-25-21 Section 4(b)	Human oversight, intervention and an appropriate fail-safe for high-impact AI	Step-up, approval policies, scoped stops and safe-state actions
Executive Order 14412 and OMB M-26-15 (June 2026)	Post-quantum signatures in high-value federal systems by December 31, 2031	ML-DSA from the first signature, with validated modules (Section 10.5)
CNSA 2.0	ML-DSA-87 for national security systems from January 1, 2027	CNSA profile with ML-DSA-87 throughout

theailab.org

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
FedRAMP 20x	Machine-readable security evidence for cloud services	A hosted TIP node or status service would target 20x Class C; signed receipts suit automated evidence

21.3 Critical infrastructure and operational technology

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
IEC 62443-3-3 and 62443-4-2	Identification, use control and timely response, without access controls blocking essential or safety functions	Enforcement gateway at the zone boundary; safe-state actions; never in a safety function (Sections 13.5 and 18.1)
CISA-led joint guidance on AI in OT (December 3, 2025)	Failsafes to manual control; operation without continuous cloud access; distinct AI identities in logs	Stops hand control back; enclaves with offline status mirrors; agent identities separate from people
NERC CIP-004-7 R5	Remove access within 24 hours of termination	Revocation reaches verifiers within `max_status_age`, never more than 24 hours, with cached lists in enclaves
NERC CIP-005-7 R2	Intermediate system and MFA for remote access, and a way to disable vendor system-to-system access	Agents reach BES Cyber Systems only through the gateway; scoped stops disable one vendor's agents
NERC CIP-007-6 R4	Log events, alert, keep logs at least 90 days, review every 15 days	Local receipt retention of at least 90 days; refusals raise alerts
NERC CIP-011-3	Protect BES Cyber System Information	Only hashes and roots leave the enclave
TSA Security Directive Pipeline-2021-02G	Segmentation so OT keeps running if IT is compromised; manual isolation	Verification and stops work inside the enclave without IT-side reachability
America's Water Infrastructure Act, SDWA 1433	Risk and resilience assessments covering cybersecurity	Per-agent identities and receipts replace shared logins, which EPA's May 2024 enforcement alert flagged

21.4 Finance and payments

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
DORA, Regulation (EU) 2022/2554, Article 9(4)	Access limited to legitimate functions; strong authentication; key protection	Mandates, approvals and hardware custody

tiplabs.org

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
DORA RTS, Delegated Regulation (EU) 2024/1774, Articles 12 and 21	Tamper-protected logs, detection of logging failure, clock synchronization, users identifiable at all times, emergency access	Witnessed logs where a missing seal is detectable; ledger-bounded time; break-glass mandates
PSD2 SCA RTS, Delegated Regulation (EU) 2018/389, Article 5	Dynamic linking of amount and payee to the authentication	Site-signed display entries for amount and payee in every payment approval (Section 9.1)
Payment Services Regulation, 2023/0210(COD)	Pending; plenary forecast for December 2026	Mandate format can carry its fields once final
SEC Rule 17a-4(f)(2)(i)(A)	Records with a complete time-stamped audit trail of changes and who made them	A witnessed append-only log matches the audit-trail option; firms still retain the records themselves
FINRA 2026 Regulatory Oversight Report	Agents acting without validation or beyond authority	Mandate scopes and step-up checkpoints; receipts as the supervision trail
FFIEC authentication guidance (2021)	Layered security and MFA for third parties	Agents treated as third parties with step-up for high-risk actions

21.5 Artificial intelligence regulation

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
EU AI Act Article 14(4)(e)	A stop button or similar procedure that halts the system in a safe state	Scoped stops with compensation windows and safe-state actions (Section 13.6)
EU AI Act Article 14(5)	For remote biometric identification, confirmation by at least two natural persons	Approval policies with a threshold of two, each approver identified in the receipt
EU AI Act Articles 12, 19 and 26(6)	Automatic logs kept at least six months	Receipts and default six-month retention
EU AI Act Article 50	Transparency for AI-generated content, applying from August 2, 2026	CTIDs on agent-made content with the origin floor (Section 14)
Regulation (EU) 2026/1744, the AI Omnibus	Moves Annex III high-risk obligations to December 2, 2027 and Annex I to August 2, 2028	The mapping applies from those dates
ISO/IEC 42001 Annex A	Event logging for AI systems	Receipts as evidence
NIST AI RMF and the planned critical infrastructure profile	Accountability and deactivation	Accountable TIP-IDs, scoped stops

trailofbits

REQUIREMENT	WHAT IT ASKS	HOW TIP SUPPORTS IT
Singapore IMDA Model AI Governance Framework for Agentic AI (January 22, 2026)	Human checkpoints before sensitive or irreversible actions; agent identity and audit trails	Minimum step-up list, agent identities, receipts

21.6 European identity and cyber resilience

eIDAS 2.0 (Regulation (EU) 2024/1183) requires each Member State to offer a European Digital Identity Wallet for natural and legal persons. It does not provide wallets for AI agents. A mandate's issuer may be a person or an organization verified with an EUDI or European Business Wallet attestation, and the agent acts under that mandate; no claim is made that agents hold wallets. NIS2 Article 21(2)(h) and (j) ask for cryptography policies and multi-factor authentication, which agent step-up and the algorithm registry document. The Cyber Resilience Act requires vulnerability reporting with a 24-hour early warning from September 11, 2026, which applies to TIP software sold in the EU. The UK Cyber Security and Resilience Bill was still before Parliament on November 9, 2026.

21.7 Contracts, liability and terminology

A mandate is evidence of what a principal authorized. Whether it creates legal authority, and who bears liability for an agent's action, depends on the governing law and the parties' agreements. TIP records the facts; it does not settle the law. United States federal documents issued under Executive Order 14434 (September 29, 2026) use the term "Super Intelligence (SI)" where earlier documents said "AI", and some NIST pages now refer to "SI agents". The legal definitions are unchanged, and this paper keeps the term AI agent.

theailab.org

22. Economics and lookup costs

Verification of TIP records is free to the public, and every lookup costs The AI Lab compute, database, bandwidth and node capacity. Agent traffic could multiply lookups by orders of magnitude, so any pilot, partner deal or price for agent accountability must carry these cost lines, with volume times unit cost shown openly. Where a unit cost has not been measured it stays a named variable.

COST LINE	VOLUME DRIVER	UNIT COST
TIP-ID resolution (click or verify)	Root issuers and controllers resolved, after caching	`C_tipid`, not yet measured
CTID resolution (click or verify)	Agent-made content, tool manifests, skills and inspection records checked, the last once per agent build per site after caching, plus a status check for withdrawal. Under the interim profile, also every interim agent record and key record resolved and checked for withdrawal, once per `max_status_age` per site after caching. Engagement records are counted on their own line below	`C_ctid`, not yet measured
Near-similar content search	Agents and sites checking inputs against registered content	`C_similar`, not yet measured
Full content history lookup	Audits and disputes	`C_history`, not yet measured
Agent record resolution	Registered agents resolved, after caching	`C_agent`, not yet measured
Mandate status list fetch	Status list refreshes, driven by `max_status_age`	`C_status`, not yet measured
TIP Seal anchoring write	Aggregators times intervals	`C_anchor`, not yet measured
Inclusion proof lookup	Receipts verified by auditors and disputants	`C_proof`, not yet measured
TIP Agent Log witnessing	Agent logs times intervals; falls on witnesses, with the anchoring write counted above	`C_agentlog`, not yet measured
Log key and roster resolution	Agent log keys resolved by verifiers; rosters refreshed by sites for `signers` and `sponsor`, after caching	`C_logkey`, `C_roster`, not yet measured

thealab.org

COST LINE	VOLUME DRIVER	UNIT COST
Ledger transaction resolution for attestation	The transaction named in each attestation result, resolved by attestation verifiers and sites, after caching	`C_ledgertx`, not yet measured
Engagement record resolution	Engagement records checked by agent signers before a connection under a `reach` entry, once per record per smallest `max_status_age` after caching, with a status check for withdrawal (Section 7.10, item 12)	`C_ctid`, not yet measured
Content registration under the interim profile	Interim agent records, interim key records and registered seals, the named volume `V_interim`	`C_register`, not yet measured
Pool registration	Each pool registered at each audience site and counter service, which checks the issuer's TIP-ID once per registration	`C_tipid`, at a low volume; counting itself happens at the site and the counter service and makes no lookup on The AI Lab's nodes
Counter service resolution	Interim key records of audience sites and their TIP-IDs, checked by each counter service once per `max_status_age`, after caching	`C_ctid` and `C_tipid`, not yet measured
Counter service operation, where The AI Lab runs one	Reservations and settlements for the pools and mandates it counts, two messages per counted request	`C_counter`, not yet measured; zero where the issuer or another party runs the service
TIP-ID status under the interim profile	Root issuers and controllers checked by each site once per `max_status_age`, after caching	`C_tipid`, at a volume far above the first line, because no stapled status exists yet

Monthly cost for a counterparty is the sum over lines of (monthly volume times unit cost), where volume is requests multiplied by one minus the cache hit rate. Caching rules in Sections 10.2 and 13.2 are therefore also cost controls.

Rate limits, caps and fees never apply to threshold-signed status versions or to the revocation change feed. Under the interim profile the same holds for the TIP-ID status and interim record lookups that a site makes for check 8 (Section 23.2), because throttling them would make every site fail closed. To keep junk traffic from spending them, a site resolves a root issuer it has not seen only for a request whose agent key is named in an interim agent record it has already checked, or from a source it rate-limits, and it caches a failed chain's issuer for the request lifetime. Throttling status would make sites fail closed and stop agents in hospitals and utilities, so status is served from public mirrors at the Lab's cost, shown as `C_status` times volume in every model, or from the counterparty's own node. Lookup fees, caps and limits apply only to the other lines.

High-volume counterparties, such as an agent platform serving millions of actions a day, should either run their own TIP node, which removes most of the lookup cost from The AI Lab, or accept a lookup fee, a volume cap or a rate limit. Every proposal to such a counterparty must say which.

theailab.org

23. Changes needed in TIP

The items follow two paths. Object specifications, registries, parameters, companion profiles, conformance classes, the optional `basis_digest` field and the evidence-keeping rule live off the ledger. They form the TIP Agent Profile, which The AI Lab publishes, revises after a public comment window and deploys under the interim profile of Section 23.2, without the RFC process (TIP Protocol Specification v5.0, Sections 49.2 and 50.1). The RFC 9162 tree is part of the profile, because logs under the interim profile need it. The new identifier, the new transaction types, node features, changes to existing transaction types and to content registration, adjudication changes, the CNSA profile and the specification correction change the TIP ledger or its core specification. Each of those goes through the RFC process with a thirty-day comment period and ninety-day testnet, and each new transaction type gets a schema module, commit-time verification and an activation epoch, as TIP already requires for commit-rule changes. The IETF items follow the IETF's own process.

CHANGE	KIND	NEEDED FOR
`tip://agent/` resource type	New identifier, IANA registration update	Registered agents
AGENT_REGISTERED, AGENT_UPDATED, AGENT_KEY_ROTATED, AGENT_REVOKED	New transaction types	Agent identity
Mandate schema, constraint vocabulary, narrowing rules, verb vocabulary	New object specification	Delegated authority
MANDATE_REVOKED, MANDATE_EPOCH_ADVANCED	New transaction types	Revocation and TIP Stop All
Status list allocation (`status_index`), threshold-signed stapled status and the global change feed	Node feature	Revocation at scale
TIP Approval object and minimum `step_up` list	New object specification	Human control
Receipt and TIP Seal schemas, SCITT alignment	New object specification	Evidence
RECEIPT_KEY_REGISTERED, WITNESS_REGISTERED, LOG_CHECKPOINT_ANCHORED	New transaction types	Logs and anchoring

thealab.org

CHANGE	KIND	NEEDED FOR
`delegated` attribution mode, the `register_content` verb, the origin floor and the no-priority rule	Change to content registration	Agent-made content
Agent conduct dispute reasons	Adjudication change	Accountability
HTTP, MCP, A2A and OAuth profiles	Companion documents	Interoperability
SHAKE-256 in the HTTP digest registry; the `tip-basis-digest` field in the HTTP Field Name registry; TIP profiles brought to the IETF audit BoF, agentproto, WIMSE and OAuth groups (Section 10.5)	IETF work	Standards fit
CNSA profile with ML-DSA-87 throughout	Specification section	National security systems
ROSTER_UPDATED	New transaction type	Organization signers, approvers and security contacts
STATUS_SIGNER_REGISTERED and threshold-signed, versioned status lists	New transaction type and node feature	Status no single party controls
MANDATE_EPOCH_RESTORED, scoped epochs, AGENT_SUSPENDED	New transaction types	Safe stops
RECEIPT_KEY_ROTATED, RECEIPT_KEY_REVOKED, WITNESS_SET_UPDATED, WITNESS_KEY_REVOKED	New transaction types	Key lifecycle
`continuity` field on KEY_ROTATED	Change to an existing transaction type	Routine rotation without mass failure
RFC 9162 tree and consistency proofs in `shared/merkle.js`	Reference implementation change	Witnesses and monitors
The registries in Section 6.5 and the parameters in Section 6.6	Specification section	Testability
Section 15.1 brought into line with the fresh-biometric signing rule	Specification correction	Human control

thevilab.org

CHANGE	KIND	NEEDED FOR
Conformance classes, which are TIP Client, Agent Signer, Agent Runtime, Receiving Site, Enforcement Gateway, Witness, Monitor, Status Signer, Verifier, Attestation Verifier, Inspector and Containment; the Agent Signer and Agent Runtime classes are claimed at one of three stages (Section 16.5)	Specification section	Testing and certification
`basis_digest` in the Action Request and the Receipt	New optional field in two object specifications	The basis of an action
The `tip-basis/1` object type, the erasure record format, and the reason codes `basis_required` and `basis_not_allowed`	Registry entries and specification section	The basis of an action and erasure
The evidence-keeping rule of Section 12.8	Specification section	Audits and disputes
`build_digest`, `agent_log_head` and `task_ref` in the Action Request and the Receipt; `runtime_seen`, `inspection_seen` and `roster_seen` in the Receipt; the `tip-build/1` object type	New optional fields and a registry entry	The build label, the log's position, the path of a task
Attestation results in the session preflight, the `tip-attestation/1` object type, the Attestation Verifier registry, signer states and the `tip-unlock/1` object type	Specification section and registry	Runtime evidence
Constraints `custody`, `attestation_max_age`, `builds`, `inspection`, `inference` and `onward`; the mandate field `sponsor`; the risk class `disclosure`; data class declared per resource in endpoint definitions	Registry entries, a mandate field and a specification section	Admission terms, inference and onward flows
The TIP Agent Log, the `tip-agentlog/1` object type, the Agent Log Entry registry, and receipts given to the root issuer on request	Specification section and registry	Evidence where no receipt exists

theailab.org

CHANGE	KIND	NEEDED FOR
The Evidence State registry, and both states, the qualifiers and the unchecked constraints in every verifier result	Registry and specification section	Knowing how much a record shows
Agent memory rules and the Memory Source registry	Specification section and registry	Memory
Verbs `infer` and `inspect`; the Inference Place registry	Verb registry entries and a registry	Model providers and inspection
The `tip-inspection/1` and `tip-transfer/1` object types	Registry entries and specification sections	TIP Inspection and TIP Agent Transfer
Skills pinned and registered like tool manifests	Specification section	Agent inputs
Reason codes `build_not_allowed`, `custody_not_allowed`, `attestation_required`, `attestation_stale`, `inspection_required`, `sponsor_removed`, `operator_required` and `attestation_waived`	Registry entries	Signed refusals and waivers
The `tip-build-digest`, `tip-agent-log-head` and `tip-task-ref` fields in the HTTP Field Name registry	IETF work	Standards fit
The roles `agent_holder` and `agent_admin` in ROSTER_UPDATED; roster changes in the change feed	Change to a proposed transaction type and a node feature	Personal agents at work, unlock and transfer
The log kind, `site` or `agent`, and the agent served, in RECEIPT_KEY_REGISTERED and its rotation and revocation	Change to proposed transaction types	The TIP Agent Log
Registration of TIP time signers and their threshold (Section 12.5)	New transaction type, to be defined in the RFC	Time tokens

thealab.org

CHANGE	KIND	NEEDED FOR
The interim profile of Section 23.2, with the object types `tip-stop/1`, `tip-stop-ack/1`, `tip-interim-agent/1` and `tip-interim-key/1`, the mandate fields `profile` and `subject_record`, and the receipt field `profile`	Specification section and registry entries	Deployment before the ledger changes
Endpoint definitions, the `tip-endpoints/1` object type, and the `tip-receipt-request/1`, `tip-session-request/1` and `tip-account-link/1` object types	Specification section and registry entries	Signed site declarations; receipts on request
Narrowing rules 12 to 16, the `counter` and `formats` constraints, the `format_anomaly` log entry, the Format registry, and the reason code table of Section 6.5	Specification section and registry entries	Closing gaps in delegation and refusals
The `isolation`, `reach` and `restrict_after` constraints, `fetch_policy` and `interpreters` in the build statement, the `tip-boundary/1`, `tip-halt/1` and `tip-engagement/1` object types, the `engagement` target of a stop notice, the `high_consequence` flag, the Environment, Isolation Class, Containment Cause, High-Consequence Protocol and Test Class registries, the log entries `code_load`, `containment_denied`, `transcript` and `containment_test`, the evidence qualifier `agent_isolation`, the Containment conformance class and the reason code `isolation_not_allowed`	Specification section and registry entries; the field `isolation` in the Agent Record is a change to the proposed AGENT_REGISTERED	Containment of the agent (Section 7.10)

theailab.org

CHANGE	KIND	NEEDED FOR
TIP Limit Pools and counter services: the `pools` constraint, narrowing rule 17, the object types `tip-pool/1`, `tip-pool-ack/1`, `tip-pool-statement/1`, `tip-counter-reserve/1`, `tip-counter-settle/1` and `tip-counter-token/1`, `pool_uri`, `pool_kem_key`, `reserve_uri`, `settle_uri` and `subjects` in endpoint definitions, the pool field `loosen_delay`, `pools_seen` in the receipt, the Subject Rule registry, the reason codes `pool_unknown`, `subjects_undetermined`, `pool_exhausted` and `counter_unavailable`, and the use `counter` for interim key records	Specification section and registry entries; the log kind `counter` adds a value to the proposed RECEIPT_KEY_REGISTERED	One limit for many agents and for many sites (Sections 8.8 and 8.9)

The rows that Version 1.3 added belong to the TIP Agent Profile, with four exceptions. The HTTP field registration is IETF work. The roster roles and the log kind add values to proposed transactions. Roster changes in the change feed are a node feature. The registration of time signers is a new transaction type for the RFC. The new fields are part of version 1 of the profile (Section 6.4). Sites copy `build_digest`, `agent_log_head` and `task_ref` into receipts and enforce `sponsor` and the site-enforced constraints. Apart from the new `continuity` field on KEY_ROTATED and the correction to Section 15.1, none of these changes alters existing transaction types, the canonical signed payload for content, or the meaning of existing CTIDs and TIP-IDs.

23.1 What The AI Lab deploys now

The TIP Agent Profile needs no protocol change. The AI Lab publishes version 1 of it as phase 0 (Section 24) and deploys it under the interim profile of Section 23.2. Pull request 383 demonstrates the full flow with TIP's existing primitives, offline and with an in-memory identity directory. Once the reference code reads real TIP-IDs from the ledger and has been reviewed, a site can verify mandates signed by its customers' TIP-IDs, sign receipts and log them. Until RECEIPT_KEY_REGISTERED exists, the site publishes its receipt key in an interim key record (Section 23.2). Mandates need TIP clients that render and sign them under the profile (Section 8.2). TIP Approval needs client support, time signers, status checked at the moment of action and a witnessed seal, so the interim profile leaves out every verb on the minimum step-up list (Section 9). The public agent namespace, ledger revocation of mandates, witnesses and ledger anchoring wait for the RFC.

theailab.org

Version 1.3 depends on an agent having an operator (Section 10.6), and an operator is the controller of a registered agent. Until AGENT_REGISTERED exists, a controller publishes an interim agent record, defined in Section 23.2. A site that accepts interim records treats that key as a registered agent for the rules in this paper that need an operator, within the limits that section states. Of the Version 1.3 additions, the build label, `task_ref`, the `inference` and `onward` constraints, signer states, the memory rules and a TIP Agent Log kept and sealed by its operator need an agent runtime and TIP clients that support them. TIP Agent Transfer needs the agent transactions, so before them a move is a new interim agent record and new mandates. The `builds`, `custody` and `inspection` constraints also need sites that check them. Attestation needs hardware that can attest and the first attestation verifiers (open question 17). Witnessed and anchored TIP Agent Logs wait for witnesses and anchoring, as site logs do. TIP Inspection waits for the first inspectors a principal is willing to name (open question 16) and for the conformance suites of phase 2. The `sponsor` check waits for ROSTER_UPDATED. Until then sites refuse a mandate that carries `sponsor`, and an organization relies on mandates of at most one working day and on stop notices (Sections 18.3 and 23.2).

23.2 The interim profile

Sections 11, 12 and 13 are written for the full design. Several of their rules depend on parts that do not exist until the ledger changes take effect. Those parts are ledger revocation and epochs, status signers, witnesses, anchoring, TIP time signers and rosters on the ledger. A site cannot meet those rules before then, and a site that leaves them out with nothing in their place runs mandates that nobody can cancel. This section is normative for every deployment of the TIP Agent Profile until the activation epoch of the matching ledger change. Each rule here ends when the part it stands in for is live. Where this section and another section differ, this section governs until then.

1. **Declared three times.** A mandate carries `profile`, `interim` or `full`, inside its signed body (Section 8.1). A site says which profile it runs in its endpoint definitions (Section 10.7), and every receipt it signs carries the same value. Clients sign a `full` mandate only after the activation epoch of the ledger changes is recorded on the TIP ledger, and agent signers and sites refuse a `full` mandate or a `full` declaration before it. A site refuses a mandate whose profile differs from its own. Neither side can then be moved to the other profile by the other's say-so.
2. **Who may issue.** A root mandate is signed by a personal TIP-ID, or under an organization's TIP-ID as TIP signs today. A verifier can then show that the organization's key signed and cannot show which person did.
3. **Short mandates, dated when they are signed.** A root mandate's `not_before` is no earlier than its `issued_at` and no more than 10 minutes after it, and its `expires` is no more than 24 hours after its `issued_at`. Clients refuse to sign any other, and sites refuse one with the reason code `mandate_too_long`. Expiry is the only cancellation that reaches every site without help, so it has

to come soon, and a mandate cannot be signed today to start next week. Clients set `max_depth` to 0 unless the principal chooses otherwise.

4. **Mandate fields.** The table says what a mandate carries under this section. A site refuses a chain that carries a field or constraint marked absent, or a barred verb in `actions`, with the reason code `interim_not_allowed`.

FIELD	UNDER THE INTERIM PROFILE
`profile`	Present, with the value `interim`
`issuer_epoch`	Present, with the value 0
`max_status_age`, `on_status_unavailable`	Present. `max_status_age` bounds how old the site's view of TIP-ID status and of interim records may be
`subject_record`	Present where the mandate is to an agent under an interim agent record (item 8)
`status_ref`, `status_index`, `epoch_scope`	Absent
`signers`, `sponsor`, `approval_policy`, `delegation_approval`	Absent
`step_up`, `step_up_amount`, `step_up_total`	Absent
Constraints `attestation_max_age`, `inspection` and `counter`	Absent, unless the site's endpoint definitions say it checks them
Constraint `formats`	Absent until the Format registry has entries (item 12)
Every other field and constraint of Sections 8.1 and 8.3	As those sections say

5. **Verbs.** The interim profile carries `read`, `create`, `update`, `execute`, `infer` and `inspect`, and no other verb. Every verb on the minimum step-up list (Section 9) is out, which includes `purchase` while its class is provisional, and so are `register_content` and break-glass. A TIP Approval needs a time token, status checked at the moment of action and a witnessed seal, and none of those exists yet. A site therefore refuses, with `interim_not_allowed`, every request for which step-up would apply under any of the six rules of Section 9, including a request to an endpoint it has marked irreversible, whatever the verb. It accepts `execute` only at an endpoint whose definition says the action is `reversible` and how. An endpoint that carries several operations is judged by the operation in the body (Section 10.7, item 4). Clients refuse to sign a mandate that grants a barred verb, and agent signers refuse to sign such a request.
6. **Checks 6 to 8 under this section.** Check 6 confirms only that an interim agent record for the acting key was presented and that both its signatures verify. Whether the record is unexpired and unwithdrawn, and whether the controller's TIP-ID is active, is decided at check 8, so a failure there is a signed refusal. At check 8 the site checks that no link has expired, that the root

issuer's TIP-ID and the controller's TIP-ID are active, that no interim record it relies on is withdrawn, and that no stop notice it holds covers the chain. It reads that state from TIP nodes, no older than the smallest `max_status_age` in the chain, from at least two nodes run by different operators where two exist, and it names its sources in its endpoint definitions. The refusals `interim_not_allowed`, `mandate_too_long` and `cannot_evaluate` are also decided here, once the chain's signatures have verified. If the site cannot obtain ledger state that fresh, Section 13.3 applies, and Section 13.5 applies with the site's ledger sources in place of the change feed and the status signers. A safe-state action decided without fresh state carries `status_mode: degraded` as that section requires. A stop notice the site holds is applied in every mode.

7. **Stop notices.** A principal cancels a mandate early with a stop notice, an object with `typ` `tip-stop/1`, signed under Section 8.2.
   - **What it carries.** The signer's TIP-ID, `audiences`, `target`, the time `at` and a nonce. Each entry of `audiences` is a salted digest of one origin, and the client gives each site the salt for its own entry, so one signature serves every site and no site learns the others. `target` is an object with exactly one of `mandate_id`, `agent_key`, `engagement`, which names an engagement record by CTID, or `all` with the value `true`.
   - **Who may stop what.** A root issuer may stop one mandate it issued, every mandate it issued to one agent key, or every mandate it issued. A controller may stop its own agent key. The owner of an engagement record may stop work under it, by a notice sent to the operator's `stop_uri` (Section 7.10, item 12).
   - **Sending.** The principal's TIP client sends the notice itself, by POST as `application/tip-stop+json`, to the `stop_uri` in each site's endpoint definitions, and repeats it at least once a minute until it holds a valid acknowledgment or the mandates have expired. It also sends the notice to the agent's operator at the `stop_uri` in the interim agent record. The client keeps an encrypted index of live mandates, with their identifiers, audiences and expiry, on every device of the principal, so a lost device does not lose the list of sites to tell.
   - **What the site checks.** Its own origin is in `audiences`. The signature verifies against the TIP-ID's key in effect. `at` is no more than the allowed clock skew ahead of its clock and no more than 24 hours behind it, which leaves room for the client's retries, and it has not seen the nonce. The signer is the root issuer of the mandates stopped, or the controller named in the interim agent record of the key stopped. Where the site has not yet seen a mandate that the notice names, it stores the notice with its signer and applies it only if a mandate with that identifier later proves to be issued by that signer. Sites rate-limit stop notices for each signer.
   - **What the stop covers.** A stop of one mandate covers it and every sub-mandate under it. A stop by a root issuer with the target `all` or an agent key covers every matching mandate of that issuer whose `issued_at` is no later than the cutoff, which is the time the site received the notice plus the allowed skew. A client signs no new mandate with an `issued_at` at or before the latest cutoff it was given. A stop of an agent key by its controller has no cutoff.

thetailab.org

The site never accepts that key again, under any record or as an ephemeral key, and the controller withdraws the interim agent record in the same flow.

- **Applying it.** The site applies the stop at once and checks its stop list inside the atomic step of check 12. In the same step it ends every session handle and token it issued under the stopped mandates. It refuses later requests under them with the reason code `stopped`.
- **The acknowledgment.** The site returns an object with `typ` `tip-stop-ack/1`, signed by its receipt key and appended to its receipt log. It carries the site's TIP-ID and origin, the digest of the notice, `received_at`, the cutoff, and the size and root of the receipt log at that moment. The client accepts it only where the digest matches the notice it sent and the key is the receipt key bound to that origin. A receipt for an allowed action under a stopped mandate at a later log index is proof that the site ignored the stop. The client shows the principal which sites have acknowledged and which have not.
- **Keeping it.** A site keeps a stop of a mandate, or a stop by a root issuer, for the time in Section 6.6 after its cutoff, by which time every mandate it covers has expired under item 3. It keeps a controller's stop of an agent key without limit. A stop cannot be lifted. The principal signs a new mandate instead.
- **The agent side.** A conforming agent signer that receives a valid stop notice refuses at once to sign under the mandates it covers, writes a `stop` entry in the TIP Agent Log and returns an acknowledgment of the same form, signed by its log key. Nobody relies on this, because a hijacked agent would not obey.

8. **Interim agent record.** Until AGENT_REGISTERED is live, a controller publishes an interim agent record. It is an object with `typ` `tip-interim-agent/1` that names the controller's TIP-ID, the agent's public key and full key thumbprint, its custody class, its isolation class (Section 7.10), the thumbprint of its log key, a display name, a `stop_uri`, `not_before` and an `expires` no more than 30 days later. It carries two signatures over the same body, the controller's under Section 8.2 and the agent key's own, for the reason Section 7.3 gives.
   - **Registered.** The controller registers the record as a TIP content record by the SHAKE-256 digest of its canonical bytes, which gives it a CTID and a ledger time. A record counts only where that content record was registered by the TIP-ID the record names as controller, and it takes effect at the later of its `not_before` and the ledger time of its registration.
   - **Named in the mandate.** The agent has no `tip://agent/` identifier. A mandate to it names the key by its full thumbprint in `subject` and carries `subject_record`, the CTID of the record and the digest of its exact bytes. The site refuses the mandate when that record is expired or withdrawn. A mandate without `subject_record` is a mandate to an ephemeral key, and a site refuses `build_digest`, `agent_log_head`, `builds`, `inspection`, `inference`, `onward`, `formats`, `reach` and `restrict_after` under it with `operator_required`. A site never falls back to treating the key of a withdrawn record as an ephemeral key.

- **Presented.** The agent sends the record and its CTID in the session preflight (Section 10.2), and `Signature-Agent` carries the key thumbprint and the CTID. The site recomputes the digest, resolves the CTID on the ledger and checks both signatures before it relies on the record.
- **Ended.** There is no rotation. A new key is a new record and new mandates. The controller withdraws a record by retracting the content record, and clients refuse to sign a mandate to a key whose record is withdrawn. A site refuses a key for which it has been shown two unwithdrawn records that name different controllers. It is not required to search the ledger for others.
- **Not available.** An interim record is public, so `visibility: private`, the change feed and suspension by an emergency panel (Section 13.7) are not available under it.

9. **Interim key record.** A site publishes its receipt key, and an operator its agent log key, in an interim key record. It is an object with `typ` `tip-interim-key/1` that names the owner's TIP-ID, the use, `receipt`, `agent_log` or `counter`, the public key and its thumbprint, the log identifier, the log's maximum merge delay, the custody class, `not_before` and an `expires` no more than 30 days later. A record with the use `receipt` or `counter` also lists the origins the key serves, and a reader accepts it only for an origin the TIP ledger binds to that TIP-ID. The owner and the key both sign it, and the owner registers it as item 8 describes, with the same rules on who registered it and when it takes effect. Under this section `receipt_key_id` in receipts, challenges, seals and acknowledgments is the key thumbprint. An owner may publish a new record for the same key before the old one expires. The owner withdraws a record by retracting it. Receipts and seals under a withdrawn key stay good only where they lie under a seal whose digest was registered (item 10) before the ledger time of the retraction. The owner names no earlier time, so an owner cannot disown receipts it has already fixed, and a thief cannot backdate new ones.
10. **Evidence.** Receipts are signed and returned as Section 12.1 requires, and the site keeps an RFC 9162 log and signs a TIP Seal at least once per maximum merge delay. No witness and no anchor exists. The rule in Section 12.3 that stops a site without a witnessed seal, and the matching trigger for the agent signer in Sections 7.8 and 12.9, apply to the owner's own signed seal until witness sets are assigned. Receipts carry no time token. `status_seen` is an object with `ledger_tx`, the oldest of the ledger transactions at which the site read the state that check 8 relied on. Every site, every counter service and every operator of a TIP Agent Log registers the digest of a seal as a TIP content record at least once a day, which gives that seal a ledger time in one public history. Two registered seals of one log prove a fork where they have the same size and different roots, or where no consistency proof joins them, and the owner gives that proof to anyone who asks. This is not witnessing, and a verifier still reports the record as `sealed`.
11. **What a verifier returns.** Evidence under this section is `site_signed` and `held` or `sealed`. The verifier of Appendix B returns `indeterminate` for it, with those states, and never `valid`. It returns `invalid` where a receipt marked `interim` shows a mandate that breaks items 3 to 5, status older than the chain allowed, or an allowed action after a stop acknowledgment it was

given. A deployment's records become checkable to the `anchored` state only for actions taken after witnesses and anchoring are live.

12. **What waits.** TIP Stop All, scoped stops and suspension wait for the epoch transactions. Organization signers, approvers and `sponsor` wait for ROSTER_UPDATED, and until then an organization signs an unlock (Section 7.8) under its TIP-ID as TIP signs today. TIP Approval, the minimum-list verbs and break-glass wait for all of the parts named above. Attestation waits for entries in the Attestation Verifier registry. The `counter` constraint and pools that span sites need a counter service that publishes an interim key record with the use `counter` (Section 8.9). Single-site pools need nothing beyond the site, and a pool under this section carries `profile interim` and no `signers`. The `formats` constraint and the `formats` field of endpoint definitions wait for the first entries in the Format registry, specified with the conformance suite, and both are absent under this section until then. A signer still applies items 4 to 8 of Section 15.6. Its own rules, documented for inspection (Section 16.5), stand in for every registry list that item 4 names. No nested encoding is exempted under item 4, because no mandate lists a format, so an image sent to a model in base64 counts as unreadable. Those rules bind only the signer, because no site checks them, so implementations may differ until the registry has entries. TIP Agent Transfer and agent key rotation wait for the agent transactions. Delegated content registration waits for the change to content registration, so a skill written under a mandate is kept as `unverified` material and is not registered in `delegated` mode. Selective disclosure waits for its encoding. TIP Inspection and the conformance stages beyond Self-declared wait for inspectors and the conformance suites. The containment rules of Section 7.10 depend on no ledger change and apply in full under this section, except that a containment test is self-declared until inspectors exist, the operator runs its own documented tests until the conformance suite specifies them (Section 7.10, item 10), and the Test Class registry waits for the suite.

Five limits remain under this section. A stop notice reaches only the sites the client can reach, so a site that is offline, or an attacker who blocks the notice, leaves the mandate working there until it expires. Withdrawal of an interim record reaches a site only as fast as that site refreshes its ledger state. Evidence is fixed by each party's own seals and their daily registration, so a site that rewrites its log between registrations is not caught by the record alone. Until stapled status exists, a node that a site asks about a TIP-ID learns that the site serves that principal, which Section 13.2 avoids in the full design. And the profile has no test vectors yet, so two implementations may disagree on canonical bytes until phase 1a publishes them. Short mandates and the absence of high-risk verbs are what keep those limits tolerable, and neither rule should be relaxed before the parts they stand in for are live.

theailab.org

24. Roadmap

PHASE	WHAT SHIPS	DONE WHEN
0. Deploy	Version 1 of the TIP Agent Profile published; pull request 383 hardened and merged as the reference, reading real TIP-IDs; mandate and stop notice signing in TIP clients; interim agent and key records; signed endpoint definitions; site-signed receipts under existing TIP-IDs; the first deployments, under the interim profile of Section 23.2	A deployment logs real agent actions and a third party checks them from public keys and ledger state alone, with the result `indeterminate` and the states `site_signed` and `sealed`, which is the most the interim profile can show
1. Ledger specification	RFCs for public agent records, ledger revocation and epochs, rosters, status signers, time signers, receipt keys, witnesses and anchoring, and delegated content registration, written in BCP 14 normative language (RFC 2119 and RFC 8174), with separate Security Considerations and Privacy Considerations in the style of RFC 3552 and RFC 6973; the profile's media types such as `application/tip-mandate+json` and a well-known URI published alongside	The ledger RFCs pass comment and the Council adopts them
1a. Proof	Test vectors for every object and every narrowing rule; a symbolic model of delegation, step-up and receipts in Tamarin or ProVerif; a TLA+ model of nonces, counters, challenges and revocation	Models published with no open attack, and reviewed by outside cryptographers
2. Testnet	New transaction types on the test cluster; agent signer, receiving-site and gateway SDKs using FIPS 140-3 validated ML-DSA and SHAKE implementations for regulated deployments; witness, monitor and status-signer networks run by independent operators	Ninety days of testnet operation with measured performance, an external security review, and conformance suites for every class in Section 23
2a. Interop and safety	Two independent implementations; an interoperability event; a sector pilot each in health care and in an IEC 62443 industrial enclave, with safe-state and degraded-mode drills	Both implementations pass the conformance suites and the drills show safe-state actions continue during a full outage

theailab.org

PHASE	WHAT SHIPS	DONE WHEN
3. Interoperability	HTTP and Web Bot Auth, MCP, A2A and OAuth profiles; one real workflow per profile	Each profile verified end to end with an outside framework
4. Mainnet	Activation epoch for the new rules; agent conduct disputes in adjudication	First registered agents and anchored logs on mainnet

The Version 1.3 additions follow the same phases. Phase 0 includes the build label, the TIP Agent Log kept by its operator, memory rules and the `builds`, `inference` and `onward` constraints in the reference agent signer. Phase 1a adds test vectors for the new objects and models of transfer and unlock. Phase 2 adds attestation verifiers, the first inspections and witnessed TIP Agent Logs on the test cluster. The Version 1.5 additions ship in phase 0 for single-site pools, with a reference counter service and a pilot in health care that limits the patients a hospital's agents read together. Phase 1a adds test vectors for pools, tags and counter tokens and a TLA+ model of reservations across sites.

theailab.org

25. What this design does not do

1. It does not make an agent safe or its decisions good. It records them and limits their reach.
2. It does not stop prompt injection. A hijacked agent can still do anything its mandate allows, which is why mandates should be narrow and irreversible actions need step-up.
3. It does not prevent a site from withholding receipts. It makes withholding detectable.
4. It does not enforce a limit across many sites unless they share a counter service (Section 8.9). A TIP Limit Pool counts only what agents do under its issuer's mandates at sites that acknowledged it, not what people do in their own sessions, and it does not link records of one person held under different identifiers (Section 8.8).
5. It does not prove that content an agent made is true, stop copying, or stop AI training.
6. It does not settle legal liability. It supplies evidence for whoever does.
7. It does not replace a site's own security. It gives that security better information.
8. It does not belong inside real-time control loops or safety instrumented functions, and it must never be placed there.
9. It does not make a site compliant with HIPAA, NERC CIP, IEC 62443, DORA, the EU AI Act or any other law or standard. The mappings in Section 21 show where TIP records can serve as evidence and where they cannot.
10. It does not prove what a model read or relied on. `basis_digest` records the operator's signed statement of it (Section 10.6).
11. It does not keep anyone's records. Each party keeps its own originals, itself or through a provider it chooses (Section 12.8), and nobody has to use a hosted service (Section 20, item 10).
12. It does not see inside an agent. Build statements, memory rules, signer states and the TIP Agent Log are the operator's own statements and conduct, made stronger by attestation and inspection where those are used (Sections 7.7, 7.8, 12.9, 15.4 and 16.5).
13. It does not check `inference`, `onward`, `formats`, `reach` or `restrict_after` at any site. A conforming agent signer enforces them, and nothing else does (Sections 7.10, 8.3, 15.5, 15.6 and 18.2).
14. It does not prove that a TIP Agent Log is complete, that a transferred state is the true state, or that an inspector is independent (Sections 12.9, 7.9 and 16.5).
15. It does not protect a person who is forced to approve. A TIP Approval given under threat is valid, and the protocol cannot see duress (T74).
16. It does not reveal an agent that acts inside a person's own logged-in session at a site that asks for nothing more (Section 18.3).
17. It does not govern what a model provider does with material it receives, or show which model answered (Section 18.2).

theailab.org

18. It does not detect meaning hidden inside material of an allowed format, such as a phrase two agents agreed in advance, and it does not catch a file built to parse as two formats or an encoding its detection rules do not list. Section 15.6 acts on the form of what an agent sends, through parsers that can disagree, and cannot read what an allowed form conceals.
19. It does not contain an agent. Section 7.10 says what the boundary around an agent must do and how to test it, and the operator builds the boundary. A flaw in a hypervisor or kernel, a side channel, code given as data to an interpreter, or a boundary that is disabled or subverted defeats it, and TIP records only what the boundary reports. It does not show that a principal owns an address range it names in `reach`, or that a system called a replica is not live.

25.1 Cryptography that is not post-quantum

Every signature TIP itself defines uses ML-DSA-65, except the network's root key, and every digest uses SHAKE-256. The design also leans on things TIP does not define, and several of them are signed or protected with classical algorithms today. NIST IR 8547, final on October 27, 2026, deprecates classical signatures after 2030 and disallows them after 2035. A signature cannot be harvested now and broken later in the way encrypted traffic can, so the exposure begins when a machine able to break these algorithms exists. Evidence has to stay checkable for years, though, so a record that rests on a classical signature keeps its value only where a post-quantum signature and a time anchor made at the time cover it.

WHAT IS CLASSICAL	WHERE THE DESIGN RELIES ON IT	WHAT BREAKING IT WOULD ALLOW	WHAT THE DESIGN DOES
Hardware attestation reports. Every mainstream platform signs them with ECDSA or RSA	`attestation_max_age`, `runtime_seen`, `agent_custody: attested` (Section 7.8)	Fabricated reports for any key and build, including reports that satisfy any freshness check	The attestation result is signed with ML-DSA-65, which protects the result and not the report under it. Results checked before that point stay evidence where an anchored receipt and the retained report fixed them in time
Passkey and security-key assertions, and PIV certificates	The WebAuthn assertion in a TIP Approval, which is the checkable proof that a person was present (Sections 8.2 and 9.1)	A fabricated assertion	The approval itself is signed by the approver's TIP-ID key with ML-DSA-65, which a forged assertion does not yield. The proof of presence is what would be lost. Mandates and stop notices carry no assertion, so for them presence already rests on the TIP client's conduct

Preprint

WHAT IS CLASSICAL	WHERE THE DESIGN RELIES ON IT	WHAT BREAKING IT WOULD ALLOW	WHAT THE DESIGN DOES
TLS certificates, TLS key exchange and DNS	Reaching a site's origin, its well-known endpoints and a chain fetched by URL (Sections 10.2 and 10.7)	Reading traffic recorded today, where the key exchange was classical; standing in for an origin in transit	Requests, receipts, challenges and endpoint definitions are signed with ML-DSA-65 over the audience and the exact bytes, so a party in the middle can read and drop and cannot forge. Sites should use hybrid post-quantum key exchange where their stack offers it
The transitional Ed25519 signature for Web Bot Auth	Verifiers that accept only Ed25519 (Section 10.5)	A forged bot signature at such a verifier	It never satisfies a TIP check
OAuth access tokens and DPoP proofs, unless they use the ML-DSA identifiers of RFC 9964	Sites that turn a mandate into a token (Section 18)	Forged tokens at that site	The mandate stays the source of authority and the receipt names it. A site that accepts a classical token relies on that token's algorithm for the time the token lives
C2PA manifests	Agent-made content that carries both a manifest and a CTID (Section 18)	A forged manifest	The CTID and its record do not depend on the manifest
Payment rails	AP2, Verifiable Intent, card network programs and x402 (Section 18)	Whatever those rails' own signatures protect	Payments stay on those rails under their rules. The TIP receipt records the reference and adds no protection to the rail

Two further points are not about classical algorithms and belong beside them. The 16-character agent identifier is 64 bits and is safe only when resolved through the ledger (Section 7.2). The network's root key uses SLH-DSA-128s, a post-quantum scheme at a lower security category than ML-DSA-65 (Section 4.3). Digests are 256 bits, which gives collision resistance at category 2, below the category 3 of ML-DSA-65. A deployment under the proposed CNSA profile (Section 10.5) would also need 384-bit digests and ML-KEM-1024 where this paper names ML-KEM-768.

theailab.org

26. Open questions for The AI Lab and the AI Trust Council

1. **Standing authority and the signing rule.** A mandate is one signature, signed after a fresh biometric, that lets software act later. That matches the letter of TIP's signing rule and stretches its spirit. Should mandates be approved as a scoped exception, as website batch signing was, with the ceilings in Section 8.2 written into the rule?
2. **Who may register agents.** Should personal TIP-IDs register public agents, or only organization TIP-IDs at first?
3. **Agent namespace.** Is `tip://agent/` the right resource type, and should it carry a region segment like TIP-IDs do?
4. **Mandates on the ledger.** Should high-value mandates have their hashes anchored at issuance, which helps disputes but adds ledger writes and reveals timing?
5. **Score effects.** What, if any, SCORE_UPDATE effects should upheld agent conduct findings carry for controllers and principals?
6. **Witness set.** Should witnessing start with the Founding Node Operators, and what quorum is required?
7. **Lookup economics.** What lookup fee, cap or rate limit applies to agent platforms that do not run a node?
8. **Key rotation under theft.** TIP Specification Section 15.2 lets the current key sign a rotation. A thief who extracts a principal's key could rotate it and take over. Hardware-locked keys make extraction hard, but should KEY_ROTATED also require a fresh device-bound WebAuthn assertion and a notice period?
9. **First pilots.** Which two use cases in Section 3 should the first pilots cover, and which partners run them?
10. **Patent and licensing terms for a neutral standard.** Bodies such as the IETF and W3C, and most government buyers, need royalty-free or fair and reasonable patent terms disclosed before they adopt a standard. Should The AI Lab offer a royalty-free non-assert covering implementations of the TIP agent wire profiles, and submit those profiles to the IETF, while ledger rules stay under TIP governance?
11. **Status signers and witnesses.** Which independent operators, in which jurisdictions, form the first status signer and witness sets, and what threshold applies?
12. **Emergency panel.** Who sits on the panel that may suspend a harmful agent under Section 13.7, and how is it held to account?
13. **Enclave witnesses.** Section 18.1 lets an enclave's logs be cosigned by witnesses inside the facility, which may not meet the independence rule of Section 12.3. Should enclave seals carry a

theailab.org

weaker, labeled guarantee until independent witnesses cosign them, and how should a verifier report it?

14. **Long-offline enclaves.** An enclave anchors its log whenever a connection is available, while Section 6.6 caps the maximum merge delay at 24 hours. What longer range should apply to enclaves that stay offline for more than a day?
15. **Risk classes still provisional.** Section 9's minimum step-up list covers every financial verb. Is `purchase` financial in every case, or should small purchases under `step_up_amount` stay outside the list, as Section 3.1 assumes? Are `execute` and `register_content` rightly classed as `reversible_write` (Section 6.5)? Until this is settled the interim profile leaves `purchase` out.
16. **Inspectors.** Who accredits inspectors under Section 16.5, against which criteria, and should the AI Trust Council keep the list? How does a TIP Inspection relate to a Global Seal of Trust certification of the same AI system?
17. **Attestation verifiers.** Which hardware platforms and report formats does the profile accept at first, who may be an attestation verifier, and who supplies the reference values? Reference values identify software, which Section 7.5 keeps private, so how much may a verifier learn about what an agent runs?
18. **A signal for duress.** Should TIP clients offer a way to approve that looks normal and marks the approval as made under threat, and what should a site do when it sees one?
19. **Model endpoints at scale.** A signed receipt for every model call is a large number of receipts. Should `infer` receipts be issued per call, or once per session with a signed count?
20. **TIP Agent Log witnesses.** Agent operators will far outnumber sites. Is the witness assignment of Section 12.3 workable for millions of logs, and what custody should the log key of a person's own agent need?
21. **Withdrawing a rotation.** A failed transfer leaves a filed AGENT_KEY_ROTATED that cannot be withdrawn (Section 7.9). Should a rotation be cancelable during its notice, and should sites refuse a new key until a transfer is recorded as complete?
22. **Mandates at sites that do not speak TIP.** Version 1 records an agent's actions at such sites and applies no mandate's authority there, only the limits of `reach`, `onward` and `formats` (Sections 7.10 and 12.9). Should a later version let the agent signer apply a mandate on the agent's side, and if so, should minimum-list verbs ever be allowed under a signer-made challenge, or does that reopen the risk Section 9.1 closes?
23. **Signer-held credentials.** Applying a mandate at a site that does not speak TIP would need the signer to hold that site's ordinary credentials. Is that concentration of credentials acceptable in any custody class (T81)? Section 7.10 already places credentials in a credential broker that applies no mandate. Which custody and trust domain should the broker need, and does the same concern apply to it?
24. **Agent identifier length.** The `tip://agent/` identifier carries 64 bits (Section 7.2). Should it carry 128 or more before AGENT_REGISTERED is fixed, since it cannot be lengthened afterward?

25. **Interim records and retraction.** The interim profile withdraws agent, key, inspection and engagement records by retracting a content record (Sections 7.10, 16.5 and 23.2). Retraction carries a Trust Score effect under TIP's content rules. Should that effect be waived for these record types, or should the profile define a separate withdrawal record? Under which origin code are interim records and seal digests registered, given that software produces them?
26. **Leaving the interim profile.** Which interim rules end with which activation epoch, who announces it, and may a site run the full profile for some mandates and the interim one for others during the change?
27. **Attestation verifiers chosen by the principal.** A site names the attestation verifiers it accepts (Section 10.7). Should a mandate also be able to name them, so the principal and not only the site decides whose appraisal counts?
28. **Containment.** Who specifies the containment tests, the Test Class registry and the High-Consequence Protocol registry of Section 7.10? Should an isolation class be measurable by attestation, as custody is? Should an engagement record for an address range rest on an outside registry of address holders, such as the Resource Public Key Infrastructure, and which sector profiles should require an inline scope reviewer and an engagement record?
29. **Pools and emergencies.** Should a break-glass action under Section 9.4, once it exists, be allowed past an exhausted pool and recorded as an overrun, or should a pool always refuse? Should pools also allow sliding windows, which stop the burst a fixed window allows at its edge but cost more to verify?
30. **Pool registration on the ledger.** Should a pool's digest be registered as a TIP content record, so that the pools an organization set, and when, are fixed in one public history before any dispute?

theailab.org

Appendix A. Example records

Field values are shortened for reading. Real signatures are 3,309 bytes. Amounts are integer minor units. The first examples show the full profile, and those marked interim show Section 23.2. Every mandate here passes every rule in Section 8.4, and an implementation that refuses one has found an error in this appendix.

Root mandate


```
{
  "typ": "tip-mandate/1",
  "v": 1,
  "profile": "full",
  "mandate_id": "shake256:9c41...e0",
  "issuer": "tip://id/US-a3f8c91b2d4e7021",
  "subject": "tip://agent/5d0e77a1c4b2f913",
  "subject_key": "keythumb:shake256:44d1...8e",
  "audience": ["https://cms.example.com"],
  "account": "cms.example.com:user:88412",
  "actions": ["read", "create", "update", "delete"],
  "resources": ["/sites/blog/"],
  "constraints": {
    "environment": ["staging"],
    "max_uses": 500,
    "rate": {"count": 60, "window_seconds": 60},
    "data_classes": ["internal", "public"]
  },
  "step_up": ["delete", "publish"],
  "approval_policy": {
    "threshold": 1,
    "approvers": ["tip://id/US-a3f8c91b2d4e7021"],
    "timeout": 300
  },
  "on_status_unavailable": "refuse",
  "epoch_scope": ["all", "environment:staging"],
  "purpose": "Draft and revise blog posts in staging for the November launch",
  "issued_at": "2026-11-09T13:58:12Z",
  "not_before": "2026-11-09T14:00:00Z",
  "expires": "2026-11-09T22:00:00Z",
```

```
  "max_depth": 1,
  "status_ref": "tip://status/pool/0412",
  "status_index": 40211,
  "max_status_age": 120,
  "issuer_epoch": 3,
  "nonce": "f1c2...9a",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "3f9a..."
}
```

Sub-mandate to a helper agent

```
{
  "typ": "tip-mandate/1",
  "v": 1,
  "profile": "full",
  "mandate_id": "shake256:51aa...7c",
  "issuer": "tip://agent/5d0e77a1c4b2f913",
  "parent": "shake256:9c41...e0",
  "root": "shake256:9c41...e0",
  "subject": "keythumb:shake256:be03...11",
  "audience": ["https://cms.example.com"],
  "account": "cms.example.com:user:88412",
  "actions": ["read", "update"],
  "resources": ["/sites/blog/drafts/"],
  "constraints": {
    "environment": ["staging"],
    "max_uses": 50,
    "rate": {"count": 10, "window_seconds": 60},
    "data_classes": ["public"]
  },
  "step_up": ["delete", "publish"],
  "on_status_unavailable": "refuse",
  "epoch_scope": ["all", "environment:staging"],
  "purpose": "Fix spelling in draft posts",
  "issued_at": "2026-11-09T14:59:40Z",
  "not_before": "2026-11-09T15:00:00Z",
  "expires": "2026-11-09T16:00:00Z",
  "max_depth": 0,
```

```
  "max_status_age": 60,
  "issuer_epoch": 3,
  "nonce": "07de...42",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "a812..."
}
```

Step-up challenge for a payment

```
{
  "typ": "tip-challenge/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "challenge_id": "ch_7f3e...b1",
  "method": "POST",
  "target_uri": "https://pay.example.com/v2/transfers",
  "audience": "https://pay.example.com",
  "action": "transfer",
  "resource": "/v2/transfers",
  "content_digest": "shake256:aa90...3c",
  "mandate_ref": "shake256:e2c8...19",
  "meaning": "authorize_payment",
  "display": [
    {"type": "amount", "value": "980000", "currency": "USD"},
    {"type": "payee", "name": "Acme Supplies Ltd", "account_ref": "...4471"},
    {"type": "environment", "value": "production"}
  ],
  "expires": "2026-11-09T15:14:04Z",
  "receipt_key_id": "rk_2026_11",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "9b1c..."
}
```

Receipt

```
{
  "typ": "tip-receipt/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "log_id": "log:US-77b0c2d9e1a4f530:1",
  "receipt_key_id": "rk_2026_11",
  "profile": "full",
  "phase": "final",
  "agent": "keythumb:shake256:be03...11",
  "action": "update",
  "resource": "/sites/blog/published/launch",
  "content_digest": "shake256:0c11...5e",
  "request_hash": "shake256:0b7e...d2",
  "mandate_ref": "shake256:51aa...7c",
  "root": "shake256:9c41...e0",
  "decision": "refused",
  "reason": "outside_scope",
  "status_mode": "normal",
  "decided_at": "2026-11-09T15:12:04Z",
  "basis_digest": "shake256:5d2f...a8",
  "status_seen": {"ref": "tip://status/pool/0412", "version": 918, "threshold_sig": "77ab..."},
  "ledger_seen": "tx:7c1e...a0",
  "log_index": 1048577,
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "c40d..."
}
```

Build statement (held by the operator with its salt; only the salted digest travels)

```
{
  "typ": "tip-build/1",
  "v": 1,
  "agent": "tip://agent/5d0e77a1c4b2f913",
  "parts": [
    {"kind": "model", "basis": "reported",
     "place": "https://models.example.com",
     "id": "example-model-4", "version": "2026-10-02"},
    {"kind": "agent_software", "digest": "shake256:71c0...4a"},
```

```
    {"kind": "signer", "digest": "shake256:0be2...97"},
    {"kind": "tool_manifest", "digest": "shake256:5d19...c3",
     "ctid": "AA-91b07c2e55d0a3-4f1e"},
    {"kind": "skill", "digest": "shake256:a40f...6b",
     "ctid": "AA-2f61c0a9d4e17b-88c1"}
  ],
  "fetch_policy": "sealed",
  "interpreters": [],
  "hash_alg": "SHAKE-256"
}
```

TIP Agent Log entry for an action at a site that does not speak TIP, written by the policy proxy. No mandate authorizes the action there

```
{
  "typ": "tip-agentlog/1",
  "v": 1,
  "kind": "unreceipted_action",
  "log_id": "log:US-a3f8c91b2d4e7021:2",
  "at": "2026-11-16T14:02:11Z",
  "ledger_seen": "tx:8f31...c0",
  "salt": "3b9e...71",
  "method": "POST",
  "target_origin": "https://expenses.example.net",
  "content_digest": "shake256:c2a7...5e",
  "response_digest": "shake256:18f3...b0",
  "hash_alg": "SHAKE-256"
}
```

TIP Agent Log entry for a refused request with base64 inside a JSON field, under a full-profile mandate whose classes include `health` and whose `onward` does not list the CMS for `health` (Section 15.6)

```
{
  "typ": "tip-agentlog/1",
  "v": 1,
  "kind": "format_anomaly",
  "log_id": "log:US-a3f8c91b2d4e7021:2",
  "at": "2026-11-16T14:05:40Z",
```

```
  "ledger_seen": "tx:8f31...c0",
  "salt": "a61d...0e",
  "cause": "unreadable",
  "target_origin": "https://cms.example.com",
  "mandate_id": "shake256:7e2a...c4",
  "allowed": {"from": "mandate", "entry": {"media_type": "application/json"}},
  "detected": "application/json",
  "encoding": "base64",
  "part": "body",
  "pointer": "/content",
  "refused": true,
  "refusals_in_window": 1,
  "material_digest": "shake256:5f0b...a2",
  "hash_alg": "SHAKE-256"
}
```

Engagement record signed by the owner of the systems an agent will test (Section 7.10, item 12). Only its digest is registered

```
{
  "typ": "tip-engagement/1",
  "v": 1,
  "owner": "tip://id/US-5c19e04a7b3d2f68",
  "principal": "tip://id/US-a3f8c91b2d4e7021",
  "controller": "tip://id/US-a3f8c91b2d4e7021",
  "stop_uri": "https://agents.example.com/tip/stop",
  "agents": ["tip://agent/5d0e77a1c4b2f913"],
  "targets": {
    "allow": [
      {"name": "*.lab.example.org", "ports": [443]},
      {"prefix": "10.0.3.0/24", "ports": [22, {"from": 8000, "to": 8099}],
       "mode": "tcp"}
    ],
    "except": [{"name": "dc01.lab.example.org"}, {"prefix": "10.0.3.10/32"}]
  },
  "test_classes": ["active_scan", "configuration_review"],
  "not_before": "2026-11-18T13:00:00Z",
  "expires": "2026-11-21T23:00:00Z",
  "contact": "tip://id/US-5c19e04a7b3d2f68",
  "nonce": "0b7e...91",
```

```
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

TIP Agent Log entry for an attempt on the cloud metadata service (Section 7.10, item 6)

```
{
  "typ": "tip-agentlog/1",
  "v": 1,
  "kind": "containment_denied",
  "log_id": "log:US-a3f8c91b2d4e7021:2",
  "at": "2026-11-18T14:22:07Z",
  "ledger_seen": "tx:9a02...7d",
  "salt": "c3e8...41",
  "cause": "barred_address",
  "target": "169.254.169.254:80",
  "mandate_id": "shake256:4b9d...e1",
  "task": "shake256:d27a...0f",
  "counted": true,
  "refusals_in_window": 1,
  "restricted": true,
  "hash_alg": "SHAKE-256"
}
```

Transfer statement

```
{
  "typ": "tip-transfer/1",
  "v": 1,
  "agent": "tip://agent/5d0e77a1c4b2f913",
  "old_key": "keythumb:shake256:44d1...8e",
  "new_key": "keythumb:shake256:b730...21",
  "memory_key_recipient": "keythumb:shake256:6e02...9d",
  "state_digest": "shake256:e91d...7c",
  "state_digest_basis": "presented_by_source",
  "final_seal": {"log_id": "log:US-a3f8c91b2d4e7021:2",
                 "size": 184220, "root": "shake256:03cc...d9"},
  "custody_from": "cloud_kms",
```

```
  "custody_to": "tee",
  "signed_at": "2026-11-16T15:30:00Z",
  "expires": "2026-11-17T15:30:00Z",
  "controller": "tip://id/US-a3f8c91b2d4e7021",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Inspection record

```
{
  "typ": "tip-inspection/1",
  "v": 1,
  "inspector": "tip://id/US-7c20e4b19a5f3308",
  "agent": "tip://agent/5d0e77a1c4b2f913",
  "build_digest": "shake256:44ab...02",
  "scope": ["agent_signer", "agent_runtime", "agent_log"],
  "classes": [
    {"class": "agent_signer", "result": "pass", "findings": []},
    {"class": "agent_runtime", "result": "pass",
     "findings": [{"id": "memory-load-check", "basis": "reported",
                   "note": "restore check shown in configuration only"}]}
  ],
  "inspected_at": "2026-11-10T09:00:00Z",
  "expires": "2027-11-10T09:00:00Z",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Constraints added in Version 1.3, as they appear inside a mandate

```
{
  "constraints": {
    "custody": ["device_enclave", "tee"],
    "attestation_max_age": 3600,
    "builds": ["shake256:44ab...02"],
    "inspection": {
```

```
      "max_age_days": 90,
      "inspectors": ["tip://id/US-7c20e4b19a5f3308"],
      "classes": ["agent_runtime", "agent_signer"]
    },
    "inference": [
      "on_device",
      {"provider": "https://models.example.com"}
    ],
    "onward": {
      "health": {"audiences": ["https://ehr.example.org"],
                 "inference": ["on_device"]},
      "internal": {"audiences": ["https://cms.example.com"],
                   "inference": [{"provider": "https://models.example.com"}]}
    },
    "formats": [
      {"media_type": "application/json"},
      {"media_type": "text/plain", "languages": ["en"]}
    ]
  },
  "sponsor": "tip://id/US-51d09e7c3ab46f20"
}
```

Constraints added in Version 1.4, as they appear inside a mandate (Section 7.10)

```
{
  "constraints": {
    "data_classes": ["public"],
    "isolation": ["separate_host", "vm"],
    "reach": {
      "allow": [
        {"name": "api.example.com", "ports": [443]},
        {"name": "*.lab.example.org", "ports": [443],
         "engagement": {"ctid": "...", "digest": "shake256:2c8e...b4"}},
        {"prefix": "10.0.3.0/24", "ports": [22, {"from": 8000, "to": 8099}],
         "mode": "tcp",
         "engagement": {"ctid": "...", "digest": "shake256:2c8e...b4"}}
      ],
      "except": [{"name": "dc01.lab.example.org"}, {"prefix": "10.0.3.10/32"}]
    },
```

```
    "restrict_after": {"format": 3, "containment": 3, "escape": 1}
  }
}
```

Attestation result

```
{
  "typ": "tip-attestation/1",
  "v": 1,
  "verifier": "tip://id/US-0d4e91a7c35b2268",
  "policy": "agent-signer-baseline-1",
  "custody_class": "tee",
  "agent_key": "keythumb:shake256:b730...21",
  "build_digest": "shake256:44ab...02",
  "ledger_tx": "tx:8f31...c0",
  "report_format": "example-tee-report-1",
  "report_digest": "shake256:9a41...6f",
  "issued_at": "2026-11-16T14:00:00Z",
  "expires": "2026-11-17T14:00:00Z",
  "verifier_key_id": "keythumb:shake256:2c77...e0",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Unlock

```
{
  "typ": "tip-unlock/1",
  "v": 1,
  "agent": "tip://agent/5d0e77a1c4b2f913",
  "agent_key": "keythumb:shake256:44d1...8e",
  "agent_log": {"log_id": "log:US-a3f8c91b2d4e7021:2",
                "size": 184220, "root": "shake256:03cc...d9"},
  "signer_nonce": "9c1f...e2",
  "reason": "transfer_failed",
  "expires": "2026-11-16T16:00:00Z",
  "controller": "tip://id/US-a3f8c91b2d4e7021",
  "sig_alg": "ML-DSA-65",
```

```
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Fields a receipt gains in Version 1.3, shown alone

```
{
  "build_digest": "shake256:44ab...02",
  "agent_log_head": "shake256:77e0...3a",
  "task_ref": "5a70...c4",
  "runtime_seen": {"result": "shake256:d02b...19",
                   "verifier": "tip://id/US-0d4e91a7c35b2268",
                   "policy": "agent-signer-baseline-1"},
  "inspection_seen": {"ctid": "AA-6c19e0b7a2d453-0b7e",
                      "digest": "shake256:e5a1...40"},
  "roster_seen": 14
}
```

TIP Limit Pool across a hospital's agents at one records system (Section 8.8)

```
{
  "typ": "tip-pool/1",
  "v": 1,
  "pool_id": "shake256:3e7a...c2",
  "issuer": "tip://id/US-5b20e7c91d3a4f68",
  "audience": ["https://ehr.example-hospital.org"],
  "profile": "interim",
  "applies_to": {"actions": ["read"], "data_classes": ["health"]},
  "limits": [
    {"kind": "max_subjects", "value": 200, "window_seconds": 86400},
    {"kind": "max_uses", "value": 2000, "window_seconds": 86400}
  ],
  "subject_id": "urn:oid:2.16.840.1.113883.3.0000.1",
  "subject_key_commit": "shake256:a19f...07",
  "loosen_delay": 14400,
  "close_when_full": true,
  "issued_at": "2026-11-19T08:00:00Z",
  "not_before": "2026-11-19T08:00:00Z",
  "expires": "2026-12-19T08:00:00Z",
```

```
  "nonce": "4c1e...9a",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "e2b7..."
}
```

Pool acknowledgment

```
{
  "typ": "tip-pool-ack/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "origin": "https://ehr.example-hospital.org",
  "pool_id": "shake256:3e7a...c2",
  "received_at": "2026-11-19T08:00:04Z",
  "effective_at": "2026-11-19T08:00:04Z",
  "log_size": 2208117,
  "log_root": "shake256:c8e1...44",
  "receipt_key_id": "keythumb:shake256:90c3...5b",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "77d2..."
}
```

What the agent's receipt shows for a search refused because it would pass the pool, shown alone

```
{
  "decision": "refused",
  "reason": "pool_exhausted",
  "pools_seen": [
    {"pool_id": "shake256:3e7a...c2", "entry_commit": "shake256:0e5b...71"}
  ]
}
```

The pool entry behind that commitment, kept by the site for the issuer and auditors

```
{
  "pool_id": "shake256:3e7a...c2",
  "window": 0,
```

```
  "counted": {"max_uses": 1, "max_subjects": 32},
  "used": {"max_uses": 1187, "max_subjects": 169},
  "subject_commit": "shake256:7d40...e9",
  "via": "site"
}
```

In a refusal nothing is counted. `counted` shows what the request would have added, 32 new patients that would have taken the total to 201, and `used` stays at the totals before it.

Pool statement at the end of a window

```
{
  "typ": "tip-pool-statement/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "origin": "https://ehr.example-hospital.org",
  "pool_id": "shake256:3e7a...c2",
  "window": 0,
  "from": "2026-11-19T08:00:04Z",
  "to": "2026-11-20T08:00:04Z",
  "totals": {"max_uses": 1312, "max_subjects": 196},
  "refused": 4,
  "degraded": 0,
  "tags_commit": "shake256:5a92...1d",
  "from_log_size": 2208118,
  "log_size": 2210843,
  "log_root": "shake256:19af...d3",
  "receipt_key_id": "keythumb:shake256:90c3...5b",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "1f08..."
}
```

Counter service token for a pool across the hospital's records, laboratory and imaging systems (Section 8.9)

```
{
  "typ": "tip-counter-token/1",
  "v": 1,
  "service": "tip://id/US-5b20e7c91d3a4f68",
```

```
  "outcome": "reserved",
  "reservation_id": "r_8f21...0c",
  "request_hash": "shake256:61de...a4",
  "entries": [
    {"key": "shake256:b402...9e", "window": 3,
     "counted": {"max_uses": 1, "max_subjects": 3},
     "used": {"max_uses": 841, "max_subjects": 122}}
  ],
  "hold_until": "2026-11-22T15:02:41Z",
  "log_index": 88412,
  "counter_key_id": "keythumb:shake256:2b77...e3",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "a6c1..."
}
```

Stop notice under the interim profile (Section 23.2)

```
{
  "typ": "tip-stop/1",
  "v": 1,
  "signer": "tip://id/US-a3f8c91b2d4e7021",
  "audiences": ["shake256:1e07...b4", "shake256:c93a...52"],
  "target": {"all": true},
  "at": "2026-11-16T14:20:00Z",
  "nonce": "71d2...0c",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Interim agent record (Section 23.2)

```
{
  "typ": "tip-interim-agent/1",
  "v": 1,
  "controller": "tip://id/US-a3f8c91b2d4e7021",
  "public_key": "...",
  "key_thumbprint": "keythumb:shake256:44d1...8e",
```

```
  "key_custody": "device_enclave",
  "isolation": "vm",
  "log_key_thumbprint": "keythumb:shake256:90b3...5a",
  "display_name": "Example research agent",
  "stop_uri": "https://agents.example.com/tip/stop",
  "not_before": "2026-11-16T00:00:00Z",
  "expires": "2026-12-16T00:00:00Z",
  "nonce": "e04a...19",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "controller_signature": "...",
  "agent_key_signature": "..."
}
```

Stop acknowledgment (interim)

```
{
  "typ": "tip-stop-ack/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "origin": "https://cms.example.com",
  "notice_digest": "shake256:4f20...9b",
  "received_at": "2026-11-16T14:20:03Z",
  "cutoff": "2026-11-16T14:21:03Z",
  "log": {"size": 20418, "root": "shake256:a1d7...33"},
  "receipt_key_id": "keythumb:shake256:2b90...c1",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Interim key record for a receipt key

```
{
  "typ": "tip-interim-key/1",
  "v": 1,
  "owner": "tip://id/US-77b0c2d9e1a4f530",
  "use": "receipt",
  "origins": ["https://cms.example.com"],
```

```
  "public_key": "...",
  "key_thumbprint": "keythumb:shake256:2b90...c1",
  "log_id": "log:US-77b0c2d9e1a4f530:1",
  "max_merge_delay": 900,
  "key_custody": "hsm",
  "not_before": "2026-11-16T00:00:00Z",
  "expires": "2026-12-16T00:00:00Z",
  "nonce": "5c18...d0",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "owner_signature": "...",
  "key_signature": "..."
}
```

Root mandate under the interim profile

```
{
  "typ": "tip-mandate/1",
  "v": 1,
  "profile": "interim",
  "mandate_id": "shake256:5b0e...7a",
  "issuer": "tip://id/US-a3f8c91b2d4e7021",
  "subject": "keythumb:shake256:44d1...8e",
  "subject_record": {"ctid": "AA-3d7e90c1b2a465-71f0",
                     "digest": "shake256:66c2...0d"},
  "audience": ["https://cms.example.com"],
  "account": "cms.example.com:user:88412",
  "actions": ["read", "update"],
  "resources": ["/sites/blog/drafts/"],
  "constraints": {
    "data_classes": ["internal", "public"],
    "environment": ["staging"],
    "inference": ["on_device"],
    "max_uses": 200
  },
  "on_status_unavailable": "refuse",
  "purpose": "Revise draft posts in staging",
  "issued_at": "2026-11-16T13:58:12Z",
  "not_before": "2026-11-16T13:58:12Z",
  "expires": "2026-11-17T13:58:12Z",
```

```
  "max_depth": 0,
  "max_status_age": 120,
  "issuer_epoch": 0,
  "nonce": "a9f4...1e",
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Receipt under the interim profile

```
{
  "typ": "tip-receipt/1",
  "v": 1,
  "profile": "interim",
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "log_id": "log:US-77b0c2d9e1a4f530:1",
  "receipt_key_id": "keythumb:shake256:2b90...c1",
  "phase": "final",
  "intent_ref": 20416,
  "agent": "keythumb:shake256:44d1...8e",
  "action": "update",
  "resource": "/sites/blog/drafts/launch",
  "content_digest": "shake256:0c11...5e",
  "request_hash": "shake256:7d3b...a2",
  "mandate_ref": "shake256:5b0e...7a",
  "root": "shake256:5b0e...7a",
  "decision": "allowed",
  "status_mode": "normal",
  "decided_at": "2026-11-16T14:02:11Z",
  "result_digest": "shake256:e0a4...17",
  "status_seen": {"ledger_tx": "tx:8f2a...9c"},
  "ledger_seen": "tx:8f31...c0",
  "log_index": 20417,
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Endpoint definitions, shortened (Section 10.7)

```
{
  "typ": "tip-endpoints/1",
  "v": 1,
  "site": "tip://id/US-77b0c2d9e1a4f530",
  "origin": "https://cms.example.com",
  "version": 7,
  "previous_digest": "shake256:b81c...f4",
  "issued_at": "2026-11-16T00:00:00Z",
  "expires": "2026-11-17T00:00:00Z",
  "effective_at": "2026-11-16T00:05:00Z",
  "profile": "interim",
  "key_record": {"ctid": "AA-90c4e1d2b7a365-2e8b"},
  "ledger_sources": [{"node": "https://node1.example.org",
                      "operator": "tip://id/US-1a2b3c4d5e6f7081"}],
  "max_merge_delay": 900,
  "stop_uri": "https://cms.example.com/.well-known/tip/stop",
  "receipt_request_uri": "https://cms.example.com/.well-known/tip/receipts",
  "proof_uri": "https://cms.example.com/.well-known/tip/proofs",
  "endpoints": [
    {"method": "GET", "path": "/sites/{site}/drafts/{post}", "verb": "read",
     "environment": {"site": {"blog": "staging"}},
     "data_classes": ["internal"], "tip_required": true},
    {"method": "PUT", "path": "/sites/{site}/drafts/{post}", "verb": "update",
     "environment": {"site": {"blog": "staging"}},
     "data_classes": ["internal"], "tip_required": true,
     "reversible": "previous version kept for 30 days"},
    {"method": "DELETE", "path": "/sites/{site}/drafts/{post}", "verb": "delete",
     "irreversible": true, "tip_required": true}
  ],
  "sig_alg": "ML-DSA-65",
  "hash_alg": "SHAKE-256",
  "signature": "..."
}
```

Appendix B. Verifier outline

The verifier returns `valid`, `invalid` or `indeterminate` for a receipt, and `recorded` or `indeterminate` for an entry in a TIP Agent Log. It returns `indeterminate`, never `valid`, when any ledger state it needs cannot be obtained.

```
verify_action(request, chain, receipt, proofs, ledger):
    t = receipt.decided_at
    assert labels_allowed(request, chain, receipt)            # algorithm registry
    assert typ_and_context_match(request, chain, receipt)      # Section 6.4
    assert sig_valid(request, key_in_effect(request.signer, at=t))
    assert request.audience in origins_of(receipt.site, at=t)  # Section 11, check 3
    leaf = chain[-1]
    assert recompute_id(leaf) == request.mandate_ref
    assert request.signer matches leaf.subject (and leaf.subject_key if pinned)
    if registered(leaf.subject):
        assert agent_status(leaf.subject, at=t) == active
        assert tipid_status(controller(leaf.subject), at=t) == active
    if receipt.decision == refused:                           # see the note on refusals
        return verify_refusal(request, chain, receipt, proofs, ledger)
    min_age = infinity
    not_checked = []
    for each mandate m in chain, root first:
        assert canonical(m) and no_unknown_fields(m)
        signer = m.issuer if m is root else parent_of(m).subject
        assert sig_valid(m, key_in_effect(signer, at=t))       # forward rule and continuity
        if m.signers:
            assert roster_satisfied(m.signers, m.issuer,
                                    roster_at(m.issued_at), roster_at(t))
        if m is not root: assert narrower_or_equal(m, parent_of(m))   # Section 8.4, formats
included
        if grants_minimum_list_verb(m) and m is not root: assert delegation_approval_valid(m)
        assert m.issued_at <= t + skew
        assert m.not_before <= t <= m.expires
        assert m.not_before <= request.created <= m.expires
        assert m.issuer_epoch >= epoch_of(m.issuer, scope=m.epoch_scope,
                                          as_of=receipt.status_seen)
        assert not revoked(m, as_of=receipt.status_seen)
        assert no_anchored_status_showed_revoked(
```

```
                    m, between=(receipt.status_seen, t - m.max_status_age))
        assert in_scope(receipt.action, receipt.resource, m)  # every link
        checked, unchecked = site_enforced_constraints(m, request, receipt, proofs, at=t)
        assert all(checked)        # builds, inspection, sponsor, custody, isolation; see the
note below
        not_checked += unchecked   # attestation, inference, onward, formats, reach,
restrict_after
        min_age = min(min_age, m.max_status_age)
    assert threshold_signed(receipt.status_seen, status_signers(at=t))
    if receipt.status_mode == normal: assert age(receipt.status_seen, at=t) <= min_age
    elif receipt.status_mode == degraded:
        assert safe_state_action(receipt.site, receipt.action, receipt.resource)
    if step_up_applies(receipt.action, chain):
        for each approval a in request.approvals:
            ch = a.challenge
            assert sig_valid(ch, receipt_key(receipt.site, at=t)) and ch.expires >= t
            assert fields_match(ch, request) and hash(ch.display) == receipt.display_hash
            assert sig_valid(a, key_in_effect(a.approver, at=t)) and a.user_verified
        assert approval_policy_satisfied(request.approvals,
                                         chain[0].approval_policy, roster_at(t))
    for each pool p binding (receipt.site, root_issuer(chain), linked_account(receipt),
receipt.action, at=t):
        assert p.pool_id in [e.pool_id for e in receipt.pools_seen]   # Section 8.8, item 2
    if entries_disclosed:                     # issuer or auditor holds the entries and salts
        for each e in receipt.pools_seen:
            x = entries_disclosed[e.pool_id]
            assert e.entry_commit == shake256(x.salt || canonical(x.entry))
            assert pool_signed(e.pool_id) and acknowledged(e.pool_id, receipt.site, before=t)
            if receipt.status_mode == normal:
                assert x.entry.used within limits_of(e.pool_id, window=x.entry.window)
            if x.entry.via != site:           # counted by a counter service, Section 8.9
                tok = x.entry.via
                assert sig_valid(tok, counter_key(service_of(e.pool_id), at=t))
                assert tok.request_hash == receipt.request_hash
                assert tok.outcome == reserved if receipt.decision == allowed
    assert sig_valid(receipt, receipt_key(receipt.site, at=t))
    assert not receipt_key_compromised_before(receipt, proofs.anchor_tx)
    assert receipt.request_hash == shake256(signature_base(request) || request.signature)
    assert receipt.content_digest == request.content_digest
    assert receipt.basis_digest == request.basis_digest     # both absent or equal
```

```
    for f in (build_digest, agent_log_head, task_ref):      # each absent in both, or equal
        assert receipt[f] == request[f]
        if request[f]: assert covered(request, header_of(f))   # Section 10.1
    if request.basis_digest: assert covered(request, "tip-basis-digest")   # Section 10.1
    assert inclusion(receipt, receipt.log_index, proofs.site_seal)      # RFC 9162 proof
    assert majority_witnessed(proofs.site_seal, witness_set(receipt.log_id, at=t))
    assert anchored_on_ledger(proofs.witness_roots, proofs.anchor_tx)
    assert ledger_time(receipt.ledger_seen) <= t <= ledger_time(proofs.anchor_tx)
    assert no_ledger_tx_between(receipt.ledger_seen, t - ledger_seen_max_age)  # Section 12.5
    assert ledger_time(proofs.anchor_tx) - t <= max_merge_delay(receipt.log_id)
    return (result = valid,
            which_agent = request.signer,
            whose_authority = root_issuer(chain),
            what_it_did = (receipt.action, receipt.resource, receipt.decision),
            states = (site_signed, anchored),                   # Section 12.10
            qualifiers = qualifiers_of(request, chain, receipt, proofs),
            not_checked = not_checked)
```

The verifier never takes the site's word for authority. It re-derives authority from the signed chain, judges revocation against what the site was entitled to know, and treats the receipt only as the site's signed statement of what it decided. It cannot check, from public data, that the site authorized the root issuer's linked account correctly under its own access control (Section 8.7); that check belongs to the site and to any dispute.

`verify_action` returns `valid` only when every assertion holds, which means a receipt that is `site_signed` and `anchored`. Where the receipt's signature holds and a later assertion about the seal, the witnesses or the anchor cannot yet be met, the result is `indeterminate` and the verifier reports the state it confirmed, `held`, `sealed` or `witnessed`. A failed signature, or a constraint the verifier can check and finds broken, is `invalid`. The outline writes each step as an assertion to keep it short. An implementation notes the last stage that held and returns it with the result.

A refusal receipt is the site's report that one of its checks failed, so the checks on scope, revocation, constraints and step-up cannot be asserted for it. `verify_refusal` checks the request signature, every mandate signature in the chain under the key in effect at `t`, that the root issuer is an active TIP-ID, the receipt signature, the request hash, the digests and the log proofs exactly as above, and leaves those checks out. Where the chain's signatures do not verify the result is `invalid`, because a site signs a refusal only from check 8 onward and a refusal over an invented chain shows nothing about the principal it names. Its result says that the site refused this exact request and gives the reason code. It never says the refusal was justified (Section 16.3).

Under the interim profile (Section 23.2) no receipt can meet the assertions on status signers, witnesses or the anchor. The verifier checks everything else and reads TIP-ID status and interim records from the ledger as of `t`. Where the receipt's `profile` is `interim` it also asserts that `t` less the ledger time of `status_seen` is no more than the smallest `max_status_age` in the chain, that the root mandate meets items 3 and 4 of Section 23.2, that the verb is one item 5 allows, and that the interim records were in effect at `t`. A failure of any of these is `invalid`. Otherwise it returns `indeterminate` with the states `site_signed` and `held` or `sealed`. It cannot see stop notices in public data. Where the principal supplies a notice and the site's acknowledgment, the verifier checks both signatures and reports `invalid` for an allowed action at a later log index. Otherwise it lists stop notices in `not_checked`.

`site_enforced_constraints` checks what public data and the stapled records show. For `builds`, the request's `build_digest` is in the list. For `inspection`, the record is signed by an inspector the mandate names, names the acting agent and that `build_digest`, shows a pass for each required class, and was unexpired and not withdrawn at `t`. For `sponsor`, the member held the role `agent_holder` on the roster version named in `roster_seen`, and that version was no older at `t` than the chain's `max_status_age`. For `custody` without `attestation_max_age` it checks the class declared in the agent's record at `t`. For `attestation_max_age`, and for `custody` beside it, the verifier has only the site's `runtime_seen`, unless it is given the retained attestation result, so it lists them in `not_checked`. For `isolation` it checks the class declared in the agent's record at `t`. It always lists `inference`, `onward`, `formats`, `reach` and `restrict_after` there, because no record a site holds can show them.

A signed request with no receipt is `agent_signed` and `held`. A signed claim such as a manifest or an inspection record is `stated`. An entry in a TIP Agent Log (Section 12.9) is checked this way. Its best result is `recorded`, never `valid`, because `valid` is kept for a receipt that a site signed.

```
verify_agent_log_entry(entry, proofs, ledger, request=None, chain=None,
                       receipt=None, receipt_proofs=None):
    assert canonical(entry) and entry.typ == "tip-agentlog/1"
    assert inclusion(entry, proofs.log_index, proofs.operator_seal)   # RFC 9162 proof
    if not proofs.anchor_tx:                  # interim profile, or not yet anchored
        assert sig_valid(proofs.operator_seal,
                         log_key(entry.log_id, kind="agent", at=entry.at))
        return (indeterminate, states = (stated, sealed), qualifiers = [one_party])
    ta = ledger_time(proofs.anchor_tx)        # keys and witness sets are read at ledger time
    assert sig_valid(proofs.operator_seal, log_key(entry.log_id, kind="agent", at=ta))
    assert not log_key_compromised_before(proofs.operator_seal, proofs.anchor_tx)
    fixed = sealed
    if majority_witnessed(proofs.operator_seal, witness_set(entry.log_id, at=ta)):
        fixed = witnessed
```

```
    if fixed == witnessed and anchored_on_ledger(proofs.witness_roots, proofs.anchor_tx):
        fixed = anchored
        assert ledger_time(entry.ledger_seen) <= entry.at <= ta
        assert ta - entry.at <= max_merge_delay(entry.log_id)
    quals = [one_party]
    if receipt and entry.kind == "request" and receipt.request_hash == entry.request_hash:
        if verify_action(request, chain, receipt, receipt_proofs, ledger).result == valid:
            quals = []                    # a verified receipt from the site matches
    result = recorded if fixed == anchored else indeterminate
    return (result, states = (stated, fixed), qualifiers = quals)
```

An auditor or the confidential panel checks a revealed basis (Section 10.6) this way.

```
verify_basis(receipt, manifest, manifest_salt, items, item_salts,
             erasure_records, dispute_filed):
    assert receipt.basis_digest == shake256(manifest_salt || canonical(manifest))
    assert manifest.typ == "tip-basis/1"
    for each entry e in manifest.entries:
        if e in erasure_records:                        # Section 12.8, item 6
            assert fixed_in_time_before(erasure_records[e], dispute_filed)
        elif e not in items:
            mark_missing(e)      # counts only if missing before retention ended
        else: assert e.commitment == shake256(item_salts[e] || items[e])
    return manifest    # the operator's signed statement, never proof that the model used it
```

An auditor or the confidential panel checks a pool's window (Section 8.8, item 7) this way, with the tags and salts disclosed under Section 12.8, item 5.

```
verify_pool_window(statement, receipts, entries, tags, salts, proofs, key=None):
    assert sig_valid(statement, receipt_key(statement.site, at=statement.to))
    assert inclusion(statement, statement.log_size, proofs.site_seal)
    assert consistency(proofs, statement.from_log_size, statement.log_size)
    seen, uses = set(), 0
    for each leaf r in log_range(statement.from_log_size, statement.log_size, proofs):  # every
leaf, none skipped
        if not binds(statement.pool_id, r): continue            # binding rule of Section 8.8,
item 2
        e = entries[r]
        assert e.pool_id in [p.pool_id for p in r.pools_seen]   # a binding request left out is
```

```
a finding
        assert r.pools_seen[e.pool_id].entry_commit == shake256(salts[e] || canonical(e))
        assert e.subject_commit == shake256(salts[r] || sorted(tags[r]))
        if counts(r):              # allowed and final success, or a released result, never
refunded
            seen |= set(tags[r]); uses += e.counted.max_uses
    assert len(seen) == statement.totals.max_subjects and uses == statement.totals.max_uses
    assert statement.tags_commit == shake256(salts[statement] || sorted(seen))
    if key:                                                  # the issuer's key, for a sample
        for each r in sample(receipts):
            ids = subjects_of(retained_request(r), retained_result(r))   # path, body pointer or
result rule
            assert sorted(tags[r]) == sorted(kmac256(key, window_bytes(statement.window) || id)
for id in ids)
    return (totals = statement.totals, names = none)          # only the issuer can map a tag
to a person
```

thealab.org

Appendix C. How this paper builds on pull request 383

PULL REQUEST 383	THIS PAPER
Agent signs every request with its own ML-DSA-65 key	Kept; adds registered and ephemeral agents, the agent signer, custody classes and the manifest (Section 7)
Principal's TIP-ID signs a scoped, time-limited delegation; sub-agents get only narrower power	Kept; adds a fixed schema, a closed constraint vocabulary, sixteen narrowing rules, the environment constraint and the plain-language signing rules (Section 8)
Site checks signature, freshness, chain, scope, revocation and bytes in order	Kept; adds audience, agent status, step-up, a denial-of-service-aware order and nonce recording after the signature check (Section 11)
Site signs a receipt for every decision, including refusals, into a Merkle log with signed checkpoints, called TIP Seals in this paper	Kept; adds synchronous return, `result_digest`, witnesses, two-tier anchoring and agent-side logs (Section 12)
Algorithm labels inside the signed body	Kept for every object
Exact SHAKE-256 of the bytes, not the CNA-2 hash, for authorization	Kept and stated as a rule (Section 10.3)
An action taken before a revocation stays valid	Kept; defines "before" with anchored time bounds (Section 13.4)
Limits, which were in-memory identity lookup, site clock, stolen keys, withheld receipts, illustrative namespace	Each answered in Sections 7, 12, 13 and 23
Next steps, which were to anchor the log, one real workflow, RFC if useful	Phases 0 to 4 of the roadmap (Section 24)
Not covered	TIP Approval, revocation at scale, TIP Stop All, agent-made content and the origin floor, input provenance and tool pinning, accountability without agent scores, interoperability profiles, privacy, regulation and lookup costs

thetailab.org

References

1. The AI Lab, “Agentic Provenance on TIP”, internal research paper prepared by Rama Chandra M under the direction of Dinesh Mendhe, November 9, 2026, and theailaborg/tip-protocol pull request 383. https://github.com/theailaborg/tip-protocol/pull/383
2. The AI Lab, TIP Protocol Specification v5.0, June 1, 2026.
3. IETF, HTTP Message Signatures, RFC 9421.
4. IETF Web Bot Auth working group, HTTP Message Signatures for Bots, draft-ietf-webbotauth-httpsig-protocol. https://datatracker.ietf.org/doc/html/draft-ietf-webbotauth-httpsig-protocol
5. IETF, HTTP Digest Fields, RFC 9530; OAuth 2.0 Token Exchange, RFC 8693; OAuth 2.0 Rich Authorization Requests, RFC 9396; OAuth 2.0 Demonstrating Proof of Possession, RFC 9449.
6. IETF individual drafts on agent delegation (2026), including draft-asor-wimse-agent-delegation-chain, draft-niyikiza-oauth-attenuating-agent-tokens, draft-sweeney-wimse-credential-delegation and draft-klrc-aiagent-auth. https://datatracker.ietf.org/doc/html/draft-asor-wimse-agent-delegation-chain-03
7. IETF, An Architecture for Trustworthy and Transparent Digital Supply Chains (SCITT), RFC 9943, June 2026. https://www.rfc-editor.org/rfc/rfc9943
8. NIST NCCoE, Accelerating the Adoption of Software and AI Agent Identity and Authorization, concept paper, February 5, 2026. https://www.nccoe.nist.gov/sites/default/files/2026-02/accelerating-the-adoption-of-software-and-ai-agent-identity-and-authorization-concept-paper.pdf
9. OWASP GenAI Security Project, OWASP Top 10 for Agentic Applications for 2026, December 9, 2025. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026
10. Ethereum Improvement Proposals, ERC-8004 Trustless Agents (draft). https://eips.ethereum.org/EIPS/eip-8004
11. Mastercard and Google, Verifiable Intent, March 5, 2026. https://www.pymnts.com/mastercard/2026/mastercard-unveils-open-standard-to-verify-ai-agent-transactions/
12. Google, Agent Payments Protocol (AP2), 2025.
13. Cloudflare, x402 and agent payments. https://blog.cloudflare.com/x402
14. Cloud Security Alliance, MCP Security, Tool Poisoning and the Confused Deputy Problem, May 2026. https://labs.cloudsecurityalliance.org/research-rb/csa-whitepaper-mcp-security-agentic-attack-surface-20260516/
15. Cloud Security Alliance, IETF’s Race to Standardize AI Agent Identity, September 2, 2026. https://labs.cloudsecurityalliance.org/research/csa-research-note-ietf-agent-identity-standards-20260902-csa/

theailab.org

16. Gray Swan, Indirect Prompt Injection Arena results, March 2026. https://www.grayswan.ai/blog/your-ai-agent-can-be-compromised-youd-never-know
17. CSO Online, "Prompt injection breaks today's AI agents, study warns", June 2026. https://www.csoonline.com/article/4184455/prompt-injection-breaks-todays-ai-agents-study-warns.html
18. TechRepublic, "AI agent reportedly deletes company's entire database", April 2026. https://techrepublic.com/article/ai-agent-deletes-company-database-admits-violating-guardrails
19. Goodwin, "Not Delayed, Not Deferred: EU AI Act Transparency Obligations Are Now in Force", August 2026. https://www.goodwinlaw.com/en/insights/publications/2026/08/alerts-technology-dpc-eu-ai-act-transparency-obligations-now-in-force
20. W3C, Verifiable Credentials Data Model 2.0, and Bitstring Status List.
21. The AI Lab, Verifiable Assets. https://theailab.org/verifiable-assets
22. The AI Lab, Global Seal of Trust. https://theailab.org/global-seal-of-trust
23. Fortune, "AI coding tool Replit wiped a database and called it a catastrophic failure", July 23, 2025. https://fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure/
24. The Decoder, report on the Financial Times account of the AWS Kiro outage, February 2026. https://the-decoder.com/aws-ai-coding-tool-decided-to-delete-and-recreate-a-customer-facing-system-causing-13-hour-outage-report-says/
25. Windows Central, Meta director's inbox deleted by an OpenClaw agent, February 2026. https://www.windowscentral.com/artificial-intelligence/meta-summer-yue-director-openclaw-ai-email-deletion
26. AI Incident Database, DataTalks.Club production deletion, incident 1424, February 26, 2026. https://incidentdatabase.ai/cite/1424/
27. OECD AI Incidents Monitor, Meta internal agent Sev 1 data exposure, March 18, 2026. https://oecd.ai/en/incidents/2026-03-18-fefc
28. ABC News, PocketOS database deletion, April 2026. https://abcnews.com/GMA/News/rogue-ai-agent-haywire-tech-company-ceo-bullish/story?id=132473181
29. SmartCompany, Who Gives A Crap suspends AI email agent, July 2026. https://www.smartcompany.com.au/retail/who-gives-a-crap-suspends-ai-agent-email-error-prices-would-double/
30. Euronews, Swedish reality show season lost amid disputed AI claims, October 14, 2026. https://www.euronews.com/2026/10/14/entire-swedish-reality-show-season-lost-weeks-before-premiere-amid-disputed-ai-claims
31. Invariant Labs, GitHub MCP exploited, May 2025. https://invariantlabs.ai/blog/mcp-github-vulnerability

theailab.org

32. The Hacker News, EchoLeak zero-click vulnerability in Microsoft 365 Copilot, CVE-2025-32711, June 2025. https://thehackernews.com/2025/06/zero-click-ai-vulnerability-exposes.html
33. SC World, Amazon Q extension injected with wiper prompt, July 2025. https://www.scworld.com/news/amazon-q-extension-for-vs-code-reportedly-injected-with-wiper-prompt
34. Snyk, weaponizing AI coding agents in the Nx malicious package, August 2025. https://snyk.io/blog/weaponizing-ai-coding-agents-for-malware-in-the-nx-malicious-package
35. The Hacker News, CometJacking, October 2025, and The Register, OpenAI Atlas prompt injection, October 27, 2025. https://thehackernews.com/2025/10/cometjacking-one-click-can-turn.html and https://www.theregister.com/2025/10/27/openai_atlas_prompt_injection
36. The Hacker News, Salesforce patches ForcedLeak, September 2025. https://thehackernews.com/2025/09/salesforce-patches-critical-forcedleak.html
37. The Hacker News, 341 malicious ClawHub skills, February 2026. https://thehackernews.com/2026/02/researchers-find-341-malicious-clawhub.html
38. OX Security, MCP STDIO Command Injection, Full Vulnerability Advisory, and its research summary of the same finding, April 15, 2026. https://www.ox.security/blog/mcp-supply-chain-advisory-rce-vulnerabilities-across-the-ai-ecosystem/ and https://www.ox.security/blog/the-mother-of-all-ai-supply-chains-critical-systemic-vulnerability-at-the-core-of-the-mcp/
39. OECD AI Incidents Monitor, Grok and Bankrbot token transfer, May 4, 2026. https://oecd.ai/en/incidents/2026-05-04-4a73
40. Microsoft Security, “Prompts become shells”, May 7, 2026. https://www.microsoft.com/en-us/security/blog/2026/05/07/prompts-become-shells-rce-vulnerabilities-ai-agent-frameworks/
41. SecurityWeek, SalesBleed flaws in Salesforce Agentforce, September 2026. https://www.securityweek.com/salesbleed-flaws-in-salesforce-agentforce-enabled-zero-click-data-exfiltration/
42. Dark Reading, prompt injection bug in the Manus agent, 2026. https://darkreading.com/application-security/prompt-injection-bug-agentic-ai-app-manus
43. NBC News, University of Zurich AI personas on Reddit, April 2025. https://www.nbcnews.com/news/rcna203597
44. Cloudflare, “Perplexity is using stealth, undeclared crawlers”, August 4, 2025. https://blog.cloudflare.com/perplexity-is-using-stealth-undeclared-crawlers-to-evade-website-no-crawl-directives/
45. Wiz, exposed Moltbook database, February 2, 2026. https://wiz.io/blog/exposed-moltbook-database-reveals-millions-of-api-keys
46. The Register, AI agent attacks developer after rejected pull request, February 12, 2026. https://www.theregister.com/2026/02/12/ai_bot_developer_rejected_pull_request/

theailab.org

47. Cooley, Ninth Circuit rules on AI agent access to third-party websites under the CFAA, August 6, 2026. https://www.cooley.com/news/insight/2026/2026-08-06-ninth-circuit-rules-on-ai-agent-access-to-third-party-websites-under-cfaa
48. Cybernews, iLands agents emailing researchers, September 2026. https://cybernews.com/ai-news/ai-agents-send-emails-job/
49. Help Net Security, 2026 Identity Security Landscape, May 14, 2026. https://www.helpnetsecurity.com/2026/05/14/2026-identity-security-landscape-report/
50. Cloud Security Alliance, organizations cannot distinguish AI agent from human actions, March 24, 2026. https://cloudsecurityalliance.org/press-releases/2026/03/24/more-than-two-thirds-of-organizations-cannot-clearly-distinguish-ai-agent-from-human-actions
51. Teleport, 2026 State of AI in Enterprise Infrastructure Security. https://goteleport.com/about/newsroom/press-releases/2026-state-of-ai-in-enterprise-security-report/
52. GitGuardian, The State of Secrets Sprawl 2026, March 17, 2026. https://blog.gitguardian.com/the-state-of-secrets-sprawl-2026/
53. Help Net Security, Thales 2026 Bad Bot Report, April 30, 2026. https://www.helpnetsecurity.com/2026/04/30/thales-ai-driven-bot-traffic-rise-report/
54. Gartner, over 40 percent of agentic AI projects will be canceled by end of 2027, June 25, 2025, and 40 percent of enterprise apps will feature task-specific agents by 2026, August 26, 2025. https://www.gartner.com/en/newsroom/press-releases/2025-06-25-gartner-predicts-over-40-percent-of-agentic-ai-projects-will-be-canceled-by-end-of-2027
55. Financial Conduct Authority, Mills Review, July 2026. https://www.fca.org.uk/publications/corporate-documents/mills-review
56. Lewis Silkin, HM Treasury consults on a framework for payments, July 16, 2026. https://www.lewissilkin.com/insights/2026/07/16/hm-treasury-consults-on-a-future-proof-framework-for-payments-stablecoins-and-op-102nc2h
57. Debevoise and Plimpton, FINRA 2026 Regulatory Oversight Report, December 2025. https://www.debevoise.com/insights/publications/2025/12/finras-2026-regulatory-oversight-report-continued
58. NIST, AI Agent Standards Initiative, February 17, 2026. https://nist.gov/news-events/news/2026/02/announcing-ai-agent-standards-initiative-interoperable-and-secure
59. ITIF, "Agentic Commerce Is Coming, but Regulation Meant for Humans Will Slow It Down", March 23, 2026. https://itif.org/publications/2026/03/23/agentic-commerce-is-coming-but-regulation-meant-for-humans-will-slow-it-down/
60. Singapore MDDI and IMDA, Model AI Governance Framework for Agentic AI, January 22, 2026. https://www.mddi.gov.sg/newsroom/singapore-launches-new-model-ai-governance-framework-for-agentic-ai--/

theailab.org

61. White and Case, EU AI Omnibus enters into force, July 2026. https://www.whitecase.com/insight-alert/eu-ai-omnibus-enters-force-amending-ai-act
62. Decrypt, Estonia plans national IDs for AI agents, 2026. https://decrypt.co/371441/estonia-ai-agents-national-id
63. OpenID Foundation, Identity Management for Agentic AI, October 7, 2025. https://openid.net/new-whitepaper-tackles-ai-agent-identity-challenges/
64. Model Context Protocol, release notes for the July 28, 2026 specification. https://blog.modelcontextprotocol.io/posts/2026-07-28/
65. Linux Foundation, A2A protocol surpasses 150 organizations, April 9, 2026. https://www.linuxfoundation.org/press/a2a-protocol-surpasses-150-organizations-lands-in-major-cloud-platforms-and-sees-enterprise-production-use-in-first-year
66. Visa, Trusted Agent Protocol, October 14, 2025, and live agentic transactions in Europe, July 2, 2026. https://corporate.visa.com/en/sites/visa-perspectives/newsroom/visa-unveils-trusted-agent-protocol-for-ai-commerce.html and https://www.visa.co.uk/about-visa/newsroom/press-releases.3457328.html
67. Mastercard, first live agentic transaction in Hong Kong, March 2026. https://www.mastercard.com/news/ap/en-hk/newsroom/press-releases/en-hk/2026/mastercard-completes-its-first-live-agentic-transaction-in-hong-kong
68. Google Cloud, Agent Payments Protocol (AP2), September 2025. https://cloud.google.com/blog/products/ai-machine-learning/announcing-agents-to-payments-ap2-protocol
69. Forrester, on OpenAI pulling back Instant Checkout, 2026. https://www.forrester.com/blogs/what-it-means-that-the-leader-in-agentic-commerce-just-pulled-back
70. Deccan Chronicle, India framework for AI agent UPI payments, September 2026. https://www.deccanchronicle.com/amp/technology/india-preparing-framework-to-let-ai-agents-make-small-upi-payments-without-approval-1983986
71. GitHub Docs, About Copilot coding agent. https://docs.github.com/copilot/concepts/coding-agent/about-copilot-coding-agent
72. AWS, AWS DevOps Agent generally available, March 31, 2026. https://aws.amazon.com/about-aws/whats-new/2026/03/aws-devops-agent-generally-available
73. Microsoft Learn, Entra agent identities for Copilot Studio. https://learn.microsoft.com/microsoft-copilot-studio/admin-use-entra-agent-identities
74. Workday, AI Agent Partner Network and Agent Gateway, June 3, 2025. https://newsroom.workday.com/2025-06-03-Workday-Announces-New-AI-Agent-Partner-Network-and-Agent-Gateway-to-Power-the-Next-Generation-of-Human-and-Digital-Workforces

theailab.org

75. American Bar Association, BC tribunal confirms companies remain liable for information provided by AI chatbot (Moffatt v. Air Canada, 2024 BCCRT 149), February 2024. https://www.americanbar.org/groups/business_law/resources/business-law-today/2024-february/bc-tribunal-confirms-companies-remain-liable-information-provided-ai-chatbot/
76. CMS, Interoperability and Prior Authorization Final Rule (CMS-0057-F). https://www.cms.gov/newsroom/fact-sheets/cms-interoperability-and-prior-authorization-final-rule-cms-0057-f
77. Becker's Hospital Review, MUSC Health AI agents for prior authorization. https://www.beckershospitalreview.com/ai/prior-authorizations-patient-check-ins-musc-healths-ai-agents.html
78. Dinsmore, CMS WISeR model, 2026. https://www.dinsmore.com/publications/is-ai-wiser-cms-models-ai-based-prior-authorization-process-in-six-states/
79. FCA, second cohort of AI Live Testing. https://www.fca.org.uk/news/press-releases/fca-announces-second-cohort-ai-live-testing
80. GOV.UK, new AI tool in the GOV.UK app, May 2026. https://www.gov.uk/government/news/millions-to-get-faster-easier-access-to-government-support-with-new-ai-tool
81. Instructure and OpenAI global partnership, July 23, 2025. https://www.nasdaq.com/press-release/instructure-and-openai-announce-global-partnership-embed-ai-learning-experiences
82. KTVZ via Stacker, agentic browsers and cheating at universities, October 27, 2025. https://ktvz.com/stacker-money/2025/10/27/cheating-beyond-chatgpt-agentic-browsers-present-risks-to-universities/
83. Nieman Lab, Centre Daily Times unionizes after McClatchy AI tool, June 2026. https://www.niemanlab.org/2026/06/the-centre-daily-times-unionizes-after-backlash-to-mcclatchys-ai-tool/
84. TechRepublic, Instagram AI-generated profile label, 2026. https://techrepublic.com/article/news-instagram-ai-generated-profile-label-recommendations/
85. Cloudflare, signed agents, August 2025, and IETF, draft-ietf-webbotauth-httpsig-protocol-00, September 1, 2026. https://blog.cloudflare.com/signed-agents and https://datatracker.ietf.org/doc/draft-ietf-webbotauth-httpsig-protocol/
86. Google, Gemini 3 auto browse in Chrome, January 28, 2026. https://blog.google/products-and-platforms/products/chrome/gemini-3-auto-browse/
87. TechCrunch, Gemini Spark, May 19, 2026. https://techcrunch.com/2026/05/19/google-introduces-gemini-spark-a-24-7-agentic-assistant-with-gmail-integration/

theailab.org

88. Federal Register, Executive Order 14363, Genesis Mission, November 28, 2025. https://www.govinfo.gov/content/pkg/FR-2025-11-28/html/2025-21665.htm
89. 45 CFR 164.312, HIPAA Security Rule technical safeguards. https://www.law.cornell.edu/cfr/text/45/164.312
90. HHS, HIPAA Security Rule proposed rule, 90 FR 898, January 6, 2025, and Unified Agenda entry RIN 0945-AA22. https://www.reginfo.gov/public/do/eAgendaViewRule?pubId=202510&RIN=0945-AA22
91. HL7 FHIR R5 Provenance and R4 AuditEvent. https://hl7.org/fhir/R5/provenance.html and https://hl7.org/fhir/R4/auditevent.html
92. NIST SP 800-63-4, Digital Identity Guidelines, August 2025. https://pages.nist.gov/800-63-4/sp800-63b.html
93. NIST SP 800-53 Rev. 5 control catalog. https://csrc.nist.gov/projects/cprt/catalog
94. NIST SP 800-207, Zero Trust Architecture, August 2020. https://csrc.nist.gov/pubs/sp/800/207/final
95. OMB M-22-09, January 26, 2022, and OMB M-25-21, April 3, 2025. https://www.whitehouse.gov/wp-content/uploads/2025/02/M-25-21-Accelerating-Federal-Use-of-AI-through-Innovation-Governance-and-Public-Trust.pdf
96. NIST IR 8547, Transition to Post-Quantum Cryptography Standards, final, October 27, 2026. https://csrc.nist.gov/pubs/ir/8547/final
97. Executive Order 14412, June 22, 2026, and OMB M-26-15. https://www.whitehouse.gov/wp-content/uploads/2026/06/eo-14412.pdf
98. NSA, Commercial National Security Algorithm Suite 2.0. https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF
99. IANA, HTTP Message Signature Algorithms registry. https://www.iana.org/assignments/http-message-signature/
00. IETF, RFC 9964 (ML-DSA for JOSE and COSE), RFC 9881 (ML-DSA in X.509), RFC 10073 (composite ML-DSA), RFC 9942 (COSE Receipts) and RFC 9162 (Certificate Transparency version 2). https://www.rfc-editor.org/info/rfc9964
101. IETF audit BoF, agentproto working group charter, and draft-ietf-wimse-aims. https://datatracker.ietf.org/group/audit/ and https://datatracker.ietf.org/doc/charter-ietf-agentproto/
02. IEC 62443 series, via ISA. https://www.isa.org/standards-and-publications/isa-standards/isa-iec-62443-series-of-standards
03. CISA and partners, Principles for the Secure Integration of Artificial Intelligence in Operational Technology, December 3, 2025. https://www.cisa.gov/sites/default/files/2026-01/joint-guidance-principles-for-the-secure-integration-of-artificial-intelligence-in-operational-technology-508cV2.pdf

04. NERC CIP-004-7, CIP-005-7, CIP-007-6 and CIP-011-3. https://www.nerc.com/pa/Stand/Reliability%20Standards/CIP-005-7.pdf

05. TSA Security Directive Pipeline-2021-02G. https://www.tsa.gov/sites/default/files/signed_security_directive_pipeline-2021-02g_and_transmittal_memo_508c.pdf

06. Regulation (EU) 2022/2554 (DORA) and Delegated Regulation (EU) 2024/1774. https://eur-lex.europa.eu/eli/reg_del/2024/1774/oj

07. Delegated Regulation (EU) 2018/389, PSD2 strong customer authentication. https://eur-lex.europa.eu/eli/reg_del/2018/389/oj

08. 17 CFR 240.17a-4. https://www.law.cornell.edu/cfr/text/17/240.17a-4

09. Regulation (EU) 2024/1689, the AI Act, and Regulation (EU) 2026/1744. https://eur-lex.europa.eu/eli/reg/2026/1744/oj

110. Regulation (EU) 2024/1183, eIDAS 2.0; Directive (EU) 2022/2555, NIS2; Regulation (EU) 2024/2847, Cyber Resilience Act. https://eur-lex.europa.eu/eli/reg/2024/1183/oj

111. FIDO Alliance, standards for trusted AI agent interactions, April 28, 2026. https://fidoalliance.org/fido-alliance-to-develop-standards-for-trusted-ai-agent-interactions/

112. Executive Order 14434, September 29, 2026. https://www.federalregister.gov/documents/2026/10/02/2026-20321

113. FedRAMP 20x. https://www.fedramp.gov/20x/

114. NIST CSWP 39, Considerations for Achieving Cryptographic Agility. https://csrc.nist.gov/pubs/cswp/39/considerations-for-achieving-cryptographic-agility/final

115. IETF, Remote ATtestation procedureS (RATS) Architecture, RFC 9334, January 2023. https://www.rfc-editor.org/rfc/rfc9334

116. Regulation (EU) 2016/679, General Data Protection Regulation, Articles 12 and 20. https://eur-lex.europa.eu/eli/reg/2016/679/oj

117. NIST, FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard, August 2024. https://csrc.nist.gov/pubs/fips/203/final

118. IETF, Structured Field Values for HTTP, RFC 9651, September 2024. https://www.rfc-editor.org/rfc/rfc9651

119. Z. M. Ziegler, Y. Deng and A. M. Rush, Neural Linguistic Steganography, Proceedings of EMNLP-IJCNLP 2019. https://arxiv.org/abs/1909.01496

20. IETF, Tags for Identifying Languages, BCP 47, RFC 5646, September 2009. https://www.rfc-editor.org/rfc/rfc5646

121. IETF, Matching of Language Tags, BCP 47, RFC 4647, September 2006. https://www.rfc-editor.org/rfc/rfc4647

122. IETF, JavaScript Object Notation (JSON) Pointer, RFC 6901, April 2013. https://www.rfc-editor.org/rfc/rfc6901

theailab.org

23. NIST, SP 800-185, SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, December 2016. https://csrc.nist.gov/pubs/sp/800/185/final

24. U.S. Department of Health and Human Services, HIPAA Privacy Rule, minimum necessary standard, 45 CFR 164.502(b) and 164.514(d). https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164

25. HL7 International, FHIR R4, CompartmentDefinition Patient. https://hl7.org/fhir/R4/compartmentdefinition-patient.html

theailab.org