Composable Trust Infrastructure for the Agent Internet: A Federated Architecture for Verifiable AI Agent Identity, Capability Discovery, and Secure Communication

Frans Moore

March 2026

Preprint | CC-BY 4.0

Abstract

As autonomous AI agents increasingly act on behalf of humans, executing transactions, accessing services, and coordinating with other agents, the absence of a unified trust infrastructure creates systemic risk. No single protocol can answer all trust questions simultaneously: *who is this agent?*, *is its runtime legitimate?*, *what can it do?*, *what is it authorized to spend?*, and *was the action actually executed?* This paper presents a composable trust architecture where six independent, open-source protocols compose into an end-to-end verifiable agent trust stack without requiring runtime coordination between implementations. The architecture spans six layers: capability discovery (agent.json), identity verification (Open Agent Trust Registry), identity resolution (DID Resolution v1.0), encrypted transport (QSP-1), execution attestation (ArkForge), and delegation governance (Agent Passport System). Each layer answers a distinct trust question with a single shared cryptographic primitive (Ed25519) [2]. We demonstrate that the composition is achieved through three structural properties: a shared key type, a shared hosting convention (well-known files), and unidirectional data flow where each layer's output is the next layer's input. The architecture draws on foundational work in capability-based security [3, 4], decentralized trust management [8], and lattice-based access control [6] to provide a practical realization of composable trust for the emerging agent economy. The architecture has been validated through a multi-party working group with seven registered issuers, unanimous spec ratifications, and cross-implementation conformance testing across TypeScript, Python, and Swift.

Keywords: AI agent trust, federated identity, composable protocols, agent verification, capability discovery, Ed25519, zero-trust architecture, multi-agent systems

---

1. Introduction

The emerging agent economy introduces a trust problem that traditional web authentication was not designed to solve. When a human user authenticates to a service, the trust chain is short: the user proves identity via credentials, and the service grants access. When an AI agent acts on behalf of a human, the trust chain lengthens: the service must verify not only the agent's identity but also its runtime legitimacy, its authorization scope, the human principal's delegation constraints, and, after execution, that the claimed action actually occurred. Recent surveys of AI agent security threats [18] and governance frameworks [19, 20] confirm that this trust gap is among the most pressing challenges in the deployment of autonomous agents.

Existing approaches to this problem fall into two categories. Monolithic platforms (proprietary agent runtimes with built-in trust) solve the problem within their walled gardens but create vendor lock-in and prevent cross-platform agent interaction. Point solutions (individual authentication protocols, capability

registries, or execution logging systems) solve one layer but leave gaps that attackers can exploit at layer boundaries. This mirrors the broader tension in trust management between centralized and decentralized approaches first identified by Blaze, Feigenbaum, and Lacy [8].

This paper presents a third approach: a composable trust architecture where independent protocols, developed by independent teams, compose into a complete trust stack through shared cryptographic conventions rather than shared governance or shared codebases. The architecture emerged from a working group (WG) of five founding projects that discovered, through independent implementation, that their protocols were already converging on the same primitives. No single entity controls the architecture or its constituent protocols; each layer is independently governed, permissionlessly extensible, and open-source.

1.1 Contributions

This paper makes three contributions:

1. **A six-layer trust model** that decomposes the agent trust problem into orthogonal questions, each answered by an independent protocol with a well-defined interface. The decomposition draws on the principle of least authority from capability-based security [3, 4] and the layered trust model from zero-trust architecture [13].
2. **Three composition properties**: shared key type (Ed25519 [2, 16]), shared hosting convention (well-known files), and unidirectional data flow, which together enable protocol composition without runtime coordination.
3. **Empirical validation** through a multi-party working group with cross-implementation conformance testing, seven registered issuers in production, and three unanimously ratified specifications.

---

2. Problem Statement

Consider a concrete scenario: a user instructs their AI agent to pay an internet bill. The agent must interact with the user's bank or payment service. The service faces six distinct trust questions, each requiring different evidence:

#	Question	Evidence Required
1	What services exist and what do they cost?	Structured capability manifest
2	Is this agent's runtime on an approved list?	Cryptographically signed registry lookup
3	Can I resolve this agent's identity to a verifiable key?	DID resolution to Ed25519 public key
4	Is the communication channel secure?	End-to-end encrypted transport
5	Did the agent actually execute what it claims?	Signed execution receipt
6	Was the agent authorized to spend this amount?	Delegation chain with constraint verification

No single protocol should attempt to answer all six questions. A monolithic trust protocol would be brittle (a vulnerability in the payment layer compromises identity), slow to evolve (all layers must release in lockstep), and politically untenable (no single organization should control all layers of agent trust). This decomposition

follows the principle articulated by Lampson [5]: access control is most effective when the mechanisms for granting, checking, and revoking authority are clearly separated.

The challenge is designing protocols that answer their respective questions independently while composing into a verifiable end-to-end chain. Prior work on distributed credential chain discovery [22] and role-based trust management [9] provides theoretical foundations, but no existing system addresses the specific composition requirements of autonomous AI agents operating across organizational boundaries.

---

3. Architecture

3.1 Layer Model

The composable trust stack consists of six layers, each implemented by an independent open-source project:

```
Layer 6: Delegation Governance  [Agent Passport System]
  "Was the agent authorized, within what constraints?"

Layer 5: Execution Attestation   [ArkForge]
  "Did the action actually happen? Proof."

Layer 4: Encrypted Transport     [QSP-1 / qntm]
  "Is the channel secure and authenticated?"

Layer 3: Identity Resolution     [DID Resolution v1.0]
  "Where is this agent's verifiable key?"

Layer 2: Identity Verification   [Open Agent Trust Registry]
  "Is this runtime on the approved list?"

Layer 1: Capability Discovery    [agent.json]
  "What can this service do, and what does it cost?"
```

Each layer's output becomes the next layer's input. An agent discovering a service (Layer 1) extracts the service's DID from the capability manifest. The DID resolves to an Ed25519 public key (Layer 3), which the agent verifies against the trust registry (Layer 2). Communication occurs over an encrypted channel authenticated by that key (Layer 4). The execution produces a signed receipt (Layer 5) within the constraints of the delegation chain (Layer 6).

This layered decomposition is analogous to the ISO/OSI network model: each layer provides a well-defined service to the layer above while depending only on the layer below. The key difference is that these layers are not coordinated by a single standards body. They emerged independently and compose through shared conventions.

3.2 Composition Properties

The six layers compose without runtime coordination due to three structural properties.

Property 1: Shared Key Type. All six protocols use Ed25519 [2] as their primary cryptographic primitive, standardized as EdDSA in RFC 8032 [16]. The Open Agent Trust Registry requires Ed25519 keys for issuer registration. DID Resolution v1.0 requires `did:web` and `did:key` [15] (both using Ed25519). QSP-1 uses Ed25519 for channel authentication. ArkForge signs execution receipts with Ed25519. The Agent Passport

System uses Ed25519 for delegation chains. agent.json Tier 3 manifests carry Ed25519 public keys in their identity block.

This convergence was not mandated by a central authority. Each project chose Ed25519 independently for its performance characteristics (sub-millisecond signing/verification), compact key size (32 bytes), and resistance to timing attacks [2]. The convergence emerged from engineering constraints, not governance. This is consistent with Miller's observation that good security primitives tend to be adopted through practical advantage rather than mandate [4].

Property 2: Shared Hosting Convention. Three well-known files on the same domain provide the data needed for Layers 1-3:

- `/.well-known/agent.json` : Capability manifest (Layer 1)
- `/.well-known/agent-trust.json` : OATR domain verification (Layer 2)
- `/.well-known/did.json` : DID document (Layer 3)

A single HTTPS fetch to a domain can retrieve capability declarations, registry verification data, and identity resolution data. The hosting convention also serves as an implicit domain binding: publishing at a well-known path proves control of the domain without requiring an external certificate authority. This follows the same domain-validation model used by the ACME protocol for automated certificate issuance [21].

Property 3: Unidirectional Data Flow. Each layer consumes the output of the layer below and produces input for the layer above. No layer requires callbacks, webhooks, or bidirectional communication with other layers. This means:

- Layers can be implemented in any language (the WG has TypeScript, Python, and Swift implementations).
- Layers can be upgraded independently (a new version of DID Resolution does not require changes to the trust registry).
- Layers can be omitted (a service that doesn't need encrypted transport can skip Layer 4 and still verify identity).

---

4. Layer Specifications

4.1 Layer 1: Capability Discovery (agent.json)

The agent.json specification (v1.3) defines a structured manifest that services publish at `/.well-known/agent.json` . The manifest declares:

- **Intents**: Actions agents can perform, with endpoints, parameters, and return types.
- **Pricing**: Per-call, per-unit, or free-tier cost structures.
- **Payment rails**: x402 micropayments, Lightning/L402, Stripe, or custom rails.
- **Identity**: Tier 3 manifests include `identity.did` (a `did:web` identifier) and an Ed25519 public key.

The manifest answers Layer 1's question ("what can this service do?") in a machine-readable format. Agents can discover services, compare pricing, and select payment rails without human intervention. The identity block bridges to Layer 3 by providing the DID that resolves to the service's verification key.

agent.json is deliberately rail-agnostic: it describes *what* payment options exist without embedding settlement logic. This separation allows new payment rails to emerge without spec changes. The specification is open-source, MIT-licensed, and extensible by any participant without requiring approval from any central authority.

4.2 Layer 2: Identity Verification (Open Agent Trust Registry)

The Open Agent Trust Registry (OATR) is an open, federated, cryptographically signed registry of trusted agent runtimes. It is designed as public infrastructure: permissionless to join, open-source, and governed by distributed consensus rather than any single entity. It answers Layer 2's question ("is this runtime legitimate?") through a signed manifest containing all registered issuers.

The registry's design draws on two traditions: the transparency properties of Certificate Transparency [17] (all entries are publicly auditable via Git history) and the permissionless registration model of the ACME protocol [21] (prove control, receive trust, no human gatekeepers).

Registration (Permissionless, Automated). Any runtime can register by proving three things:

1. A valid Ed25519 public key (32 bytes, base64url-encoded per RFC 4648 §5).
2. Proof of key ownership: a detached Ed25519 signature over the canonical message `oatr-proof-v1:{issuer_id}`.
3. Domain verification: hosting a `/.well-known/agent-trust.json` file or including `oatr_issuer_id` in an agent.json identity block.

Registration is fully automated via CI. The verification pipeline validates the key format, verifies the Ed25519 signature, fetches the domain verification file, and auto-merges on success. No human approval is required for inclusion. There are no gatekeepers, fees, or approval committees.

Verification (Local, Sub-Millisecond). Services verify agent attestations in 14 steps, all performed locally against a cached manifest:

1. Parse the incoming JWS [10] attestation.
2. Extract `iss` (issuer ID) and `kid` (key ID) from the protected header.
3. Look up the issuer in the local manifest copy. 4-10. Validate issuer status, key status, key expiry.
4. Cryptographically verify the Ed25519 signature. 12-14. Validate claims: audience binding, expiry, nonce.

No network calls are required during verification. The manifest is fetched and cached periodically (default: 15 minutes). All JSON is canonicalized per RFC 8785 [12] before signing. Verification targets sub-millisecond completion on commodity hardware.

Governance (Threshold-Signed). The registry's governance charter mandates transition to 3-of-5 threshold signing using the FROST protocol [14] over Ed25519 for revocation decisions. Five master keys are to be distributed across the founding maintainer and four independent ecosystem reviewers. The design ensures that no single party, including the original author, can unilaterally revoke an issuer. All revocations require a public GitHub Issue with documented justification and one of five enumerated reasons: key compromise, issuer compromise, policy violation, voluntary withdrawal, or governance decision. This follows Shamir's foundational insight [7] that critical secrets should require threshold cooperation rather than single-party control.

Zero-Trust Mirrors. The manifest is cryptographically signed. Anyone can host an exact mirror without compromising security, because a tampered manifest fails signature verification. Mirrors are zero-trust messengers: they distribute data but cannot forge it. Clients always verify signatures against the root-keys trust anchor. This design follows the zero-trust principle [13] that verification must be performed at every access point regardless of network location.

4.3 Layer 3: Identity Resolution (DID Resolution v1.0)

DID Resolution v1.0 (ratified unanimously by the WG) defines how agent DIDs [15] resolve to Ed25519 public keys. The spec requires `did:web` and `did:key` as mandatory methods, with `did:aps` and

`did:agentid` as recommended extensions.

The resolution output is a public key and a sender ID (Trunc16 of SHA-256 over the raw public key bytes). The sender ID serves as a compact, collision-resistant identifier that can be matched across protocols without transmitting the full key.

The spec emerged from convergent implementation: three WG projects independently implemented SHA-256 truncated fingerprints for key identification before the spec was drafted. The spec formalized what had already converged organically, providing evidence that the abstraction is natural and implementable across different codebases and programming languages.

4.4 Layer 4: Encrypted Transport (QSP-1)

QSP-1 (Quantum-Safe Protocol v1.0, ratified unanimously) provides end-to-end encrypted communication channels between agents. Channel authentication uses Ed25519 key pairs, with identity derived from the same DID resolution layer.

QSP-1's key design decision is separating channel encryption from identity verification. The protocol establishes encrypted channels; the trust registry determines whether the parties in those channels are legitimate. This separation means QSP-1 can be deployed without the registry (for private networks) or the registry can be used without QSP-1 (for services that use standard HTTPS).

4.5 Layer 5: Execution Attestation (ArkForge)

ArkForge provides cryptographically signed execution receipts, serving as proof that an agent action actually occurred. Each receipt binds:

- The agent's DID (resolved via Layer 3)
- The action performed
- The timestamp
- An Ed25519 signature from the execution environment

The receipt chain creates an immutable audit trail. When combined with agent.json's capability declaration (Layer 1) and a DID binding (Layer 3), the composition yields: capability declaration → identity verification → execution proof.

4.6 Layer 6: Delegation Governance (Agent Passport System)

The Agent Passport System (APS) formalizes the delegation chain from human principal to agent to sub-agent. APS enforces constraints across seven dimensions (scope, spend, depth, time, reputation, values, and reversibility), which Pidlisnyi [1] formalizes as a product lattice with a monotonic narrowing invariant. This formalization builds on Denning's foundational lattice model of secure information flow [6], extending it from confidentiality classes to multi-dimensional authority attenuation.

APS produces three artifacts that compose with the lower layers:

- **AuthorizationWitness**: A signed snapshot of the agent's position in the authority lattice at execution time, bound to the registry-verified runtime identity.
- **ConstraintVector**: Runtime evaluation showing which constraint dimensions passed or failed and the remaining headroom in each dimension.
- **ConstraintFailure**: Structured denial identifying exactly which dimensions blocked authorization and why.

The delegation layer depends on the identity layer: an AuthorizationWitness is only meaningful if signed by a runtime whose identity has been independently verified. The lattice is mathematically sound [1]; the identity

infrastructure makes it operationally enforceable. This composition follows the capability-based security principle [3] that authority should be unforgeable, transferable only by explicit delegation, and attenuable but never amplifiable.

---

5. Security Analysis

5.1 Threat Model

The architecture assumes a threat model consistent with NIST's zero-trust principles [13]: no component is inherently trusted, and verification occurs at every layer boundary. Any individual component may be compromised, but the composition of independent verification layers makes systemic compromise exponentially harder.

Threat	Compromised Layer	Mitigation
Rogue mirror serves tampered registry	Layer 2 (distribution)	Ed25519 signature verification against root keys [2]
Compromised root signing key	Layer 2 (governance)	Threshold signing; single key insufficient [7, 14]
Malicious issuer registers and issues false attestations	Layer 2 (registration)	Permanent Git-backed audit trail, governance-backed revocation
Attestation replay across services	Layer 2 (attestation)	Audience (`aud`) claim binding per service [10]
Attestation replay within service	Layer 2 (attestation)	Service-provided nonce + short TTL (max 1 hour)
Stale registry missing revocation	Layer 2 (caching)	`expires_at` enforcement, 15-minute SDK cache default
Agent claims false capability	Layer 1 (discovery)	Capability manifest is service-published, not agent-published
Forged execution receipt	Layer 5 (attestation)	Ed25519 signature by execution environment
Unauthorized delegation	Layer 6 (governance)	Monotonic narrowing invariant [1, 6]

5.2 Composition Security Properties

The layered architecture provides defense-in-depth properties that monolithic systems cannot:

1. **Layer isolation**: A vulnerability in the payment layer (Layer 1 pricing) does not compromise identity verification (Layer 2). This follows the principle of least privilege applied to protocol design, where each layer has access only to the cryptographic material it needs.
2. **Independent revocation**: Each layer can revoke credentials independently. A revoked registry issuer immediately invalidates all attestations regardless of the delegation layer's state.
3. **Progressive trust**: Services can adopt layers incrementally. A service that only needs identity verification deploys Layers 2-3; a service requiring full delegation governance adds Layer 6. This

graduated approach addresses the adoption barrier identified in trust and reputation system surveys [10, 11].

4. **Transparency**: All registry state is maintained in a public Git repository, providing an append-only audit log analogous to Certificate Transparency [17]. Any participant can independently verify the complete history of issuer additions, key rotations, and revocations.

---

6. Empirical Validation

6.1 Working Group Process

The architecture was validated through a multi-party working group comprising five founding projects, each independently developed and governed:

Project	Layer	Implementation Language	Governance
agent.json	Capability Discovery	TypeScript	Open-source, MIT
Open Agent Trust Registry	Identity Verification	TypeScript, Swift	Open-source, MIT, permissionless
qntm	Encrypted Transport	Python, TypeScript	Independent maintainer
ArkForge	Execution Attestation	TypeScript	Independent maintainer
Agent Passport System	Delegation Governance	TypeScript	Open-source, Apache 2.0

Two additional projects (AgentID, Agora) joined as aligning members, contributing DID resolution implementations in Python and TypeScript respectively. No project has governance authority over any other; composition is achieved through shared conventions, not shared control.

6.2 Spec Ratification

Three specifications have been ratified unanimously by the founding members:

- **QSP-1 v1.0** (Transport): Ratified by all founding members.
- **DID Resolution v1.0** (Identity Resolution): Ratified with 8/8 conformance test vectors passing across three independent implementations (TypeScript, Python, and TypeScript/Agora).
- **Entity Verification v1.0**: Ratified by founding members with cross-implementation conformance testing.

Each ratification required independent conformance testing. No member signed off without running the test vectors against their own implementation.

6.3 Production Deployment

The registry has seven registered issuers in production at time of writing, each verified through the automated CI pipeline without human intervention. One WG member (AgentID) has deployed a seven-agent sales pipeline across 12 countries using the identity infrastructure as the operational trust layer, with verified

handoffs at every pipeline step. This represents the first known production deployment where agent identity verification serves as the actual trust mechanism rather than a demonstration.

6.4 Cross-Implementation Conformance

DID Resolution v1.0 test vectors were independently verified by:

- OATR (TypeScript): 8/8 vectors
- AgentID (Python): 8/8 vectors
- Agora (TypeScript): 8/8 vectors
- ArkForge (TypeScript): confirmed via live DID document fetch

The sender ID derivation (SHA-256 Trunc16) converged independently in three implementations before the spec was written, providing evidence that the abstraction is natural and implementable. This pattern of independent convergence, where separately developed systems arrive at compatible interfaces without coordination, is a stronger indicator of design fitness than top-down standardization.

---

7. Related Work

7.1 Foundational Security Models

Dennis and Van Horn's capability-based security model [3] established the principle that authority tokens should be unforgeable and attenuable. Miller [4] extended this to distributed systems with the object-capability model, demonstrating that capabilities can be safely delegated and composed. The composable trust stack applies these principles to AI agent authorization: each layer attenuates authority (Layer 6 narrows delegation, Layer 2 restricts to registered runtimes) without any layer amplifying it.

Denning's lattice model [6] provides the mathematical foundation for hierarchical security classes with a partial ordering. Pidlisnyi [1] extends this to a product lattice over seven constraint dimensions for agent delegation. Bell and LaPadula's confidentiality model [23] established that information flow control requires formally verifiable properties, a principle this architecture applies through Ed25519 signature verification at every layer boundary.

7.2 Decentralized Trust Management

Blaze, Feigenbaum, and Lacy [8] identified trust management as a distinct security problem requiring decentralized solutions. Li, Mitchell, and Winsborough [9] developed role-based trust management frameworks for distributed credential evaluation. The composable trust stack extends this line of work to autonomous AI agents, where the trust decision must be made in real-time, without human intervention, across organizational boundaries.

7.3 Trust in Multi-Agent Systems

Jøsang, Ismail, and Boyd [10] survey computational trust and reputation mechanisms for online service provision. Sabater and Sierra [11] survey trust models specifically for multi-agent systems. The composable trust stack differs from these approaches in that it provides cryptographic verification of identity and authorization rather than statistical reputation scoring. The registry answers "is this runtime registered?" rather than "how trustworthy has this runtime been?"

7.4 Decentralized Identity

The W3C Decentralized Identifiers specification [15] and Verifiable Credentials Data Model [24] provide the standards foundation for agent identity. The DID Resolution layer (Layer 3) builds directly on these

specifications. Allen's self-sovereign identity principles [25] inform the design philosophy: agents (and their principals) should control their own identity without depending on a central identity provider.

7.5 Zero-Trust Architecture

NIST SP 800-207 [13] defines zero-trust principles: never trust, always verify; assume breach; verify explicitly. The composable trust stack operationalizes these principles for inter-agent communication: every attestation is verified against the registry (never trust), mirrors cannot compromise integrity (assume breach), and verification is performed locally at every service boundary (verify explicitly).

7.6 Threshold Cryptography

Shamir [7] established that critical secrets should be split across multiple parties. Komlo and Goldberg [14] developed FROST, a practical two-round threshold Schnorr signature protocol applicable to Ed25519. The registry's governance model mandates threshold signing for revocation decisions, ensuring that no single maintainer can unilaterally remove an issuer from the registry.

7.7 Certificate Transparency

Laurie, Langley, and Kasper [17] introduced Certificate Transparency for publicly auditable certificate logs. The OATR registry applies similar transparency principles: all issuer additions, key rotations, and revocations are recorded in a public Git repository, providing a complete and independently verifiable audit trail.

7.8 AI Agent Governance

He et al. [18] provide a comprehensive survey of security threats to AI agents, establishing the threat landscape that trust registries are designed to mitigate. Kolt [20] examines legal and governance frameworks for autonomous AI agents. OpenAI's practices for governing agentic systems [19] define baseline responsibilities for agent developers and deployers. The composable trust stack provides the technical infrastructure to implement the accountability mechanisms these governance frameworks require.

7.9 Industry Protocols

Google's Agent-to-Agent (A2A) protocol defines Agent Cards for agent discovery but does not include a root-of-trust registry for verifying agent identity. Visa's Trusted Agent Protocol addresses trust in agentic commerce but does not provide a federated, open identity layer. The composable trust stack is designed to complement these protocols by providing the identity verification layer they lack.

---

8. Conclusion

The composable trust architecture demonstrates that the agent trust problem decomposes into six orthogonal questions, each answerable by an independent protocol. Composition is achieved not through governance mandates or API contracts but through three emergent structural properties: a shared key type, a shared hosting convention, and unidirectional data flow. The architecture has been validated through unanimous spec ratifications, cross-implementation conformance testing, and production deployment.

The key insight is that composability through convergence is more robust than composability through design. When independent teams, solving different problems, converge on the same cryptographic primitive and the same hosting conventions, the resulting composition is natural rather than imposed. Each layer can evolve independently, be implemented in any language, and be adopted incrementally. These are essential properties for infrastructure that no single organization should control.

The agent internet requires trust infrastructure that is open, permissionless, and verifiable. This paper presents one path toward that infrastructure: open protocols, federated governance, cryptographic verification, and composable layers that together answer the full chain of trust questions that autonomous agents must satisfy. The protocols described here are open-source, the registry is permissionless to join, and the specifications are developed through consensus among independent implementors. The architecture's value lies not in any single component but in the composition, which is available to anyone who implements the shared conventions.

References

[1] T. Pidlisnyi, "Faceted Authority Attenuation: A Product Lattice Model for AI Agent Governance," Zenodo, 2026. DOI: 10.5281/zenodo.19260073

[2] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77-89, 2012. DOI: 10.1007/s13389-012-0027-1

[3] J. B. Dennis and E. C. Van Horn, "Programming semantics for multiprogrammed computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143-155, 1966. DOI: 10.1145/365230.365252

[4] M. S. Miller, "Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control," PhD thesis, Johns Hopkins University, 2006.

[5] B. W. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18-24, 1974. DOI: 10.1145/775265.775268

[6] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236-243, 1976. DOI: 10.1145/360051.360056

[7] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979. DOI: 10.1145/359168.359176

[8] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp. 164-173, IEEE, 1996.

[9] N. Li, J. C. Mitchell, and W. H. Winsborough, "Design of a Role-Based Trust-Management Framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 114-130, IEEE, 2002.

[10] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618-644, 2007. DOI: 10.1016/j.dss.2005.05.019

[11] J. Sabater and C. Sierra, "Review on computational trust and reputation models," *Artificial Intelligence Review*, vol. 24, no. 1, pp. 33-60, 2005. DOI: 10.1007/s10462-004-0041-5

[12] A. Rundgren, B. Jordan, and S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, IETF, June 2020. DOI: 10.17487/RFC8785

[13] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST Special Publication 800-207, August 2020. DOI: 10.6028/NIST.SP.800-207

[14] C. Komlo and I. Goldberg, "FROST: Flexible Round-Optimized Schnorr Threshold Signatures," in *Selected Areas in Cryptography (SAC 2020)*, LNCS vol. 12804, pp. 34-65, Springer, 2020.

[15] M. Sporny, D. Longley, M. Sabadello, et al., "Decentralized Identifiers (DIDs) v1.0," W3C Recommendation, July 2022. https://www.w3.org/TR/did-core/

[16] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, IETF, January 2017. DOI: 10.17487/RFC8032

[17] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962, IETF, June 2013. DOI: 10.17487/RFC6962

[18] Y. He, G. Deng, et al., "AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways," *ACM Computing Surveys*, 2024. DOI: 10.1145/3716628

[19] Y. Shavit, S. Agarwal, et al., "Practices for Governing Agentic AI Systems," OpenAI, December 2023.

[20] N. Kolt, "Governing AI Agents," *101 Notre Dame Law Review* (forthcoming, 2025). arXiv:2501.07913

[21] R. Barnes, J. Hoffman-Andrews, D. McCarney, and J. Kasten, "Automatic Certificate Management Environment (ACME)," RFC 8555, IETF, March 2019. DOI: 10.17487/RFC8555

[22] N. Li, W. H. Winsborough, and J. C. Mitchell, "Distributed Credential Chain Discovery in Trust Management," *Journal of Computer Security*, vol. 11, no. 1, pp. 35-86, 2003.

[23] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations," Technical Report MTR-2547, The MITRE Corporation, 1973.

[24] M. Sporny, D. Longley, D. Chadwick, et al., "Verifiable Credentials Data Model v1.1," W3C Recommendation, March 2022. https://www.w3.org/TR/vc-data-model/

[25] C. Allen, "The Path to Self-Sovereign Identity," 2016. https://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html

[26] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," RFC 7515, IETF, May 2015. DOI: 10.17487/RFC7515

Author Information

Frans Moore is the original author of the Open Agent Trust Registry and the agent.json specification. Both projects are open-source, permissionlessly extensible, and governed by their respective communities. Contact: https://github.com/FransDevelopment

Data Availability

All source code, specifications, and registry data referenced in this paper are publicly available under open-source licenses:

- Open Agent Trust Registry: https://github.com/FransDevelopment/open-agent-trust-registry (MIT)
- agent.json: https://github.com/FransDevelopment/agent-json (MIT)
- DID Resolution v1.0: https://github.com/corpollc/qntm/blob/main/specs/working-group/did-resolution.md

License