AgenticIdentity: A Cryptographic Trust Anchor for AI Agent Systems

Omoshola Owolabi
Researcher – AI/ML
Agentra Labs
omoshola.owolabi@agentralabs.tech

February 24, 2026

Abstract

Large language model agents increasingly take consequential actions—deploying code, sending messages, managing infrastructure—yet lack any cryptographic mechanism to prove identity, sign actions, or establish trust. We present AgenticIdentity, a cryptographic trust anchor that gives AI agents persistent, portable, mathematically verifiable identity. The system introduces three primitives: (1) *identity anchors* rooted in Ed25519 key pairs with hierarchical key derivation via HKDF-SHA256, (2) *action receipts* that produce signed, chainable, witness-attested proofs of every agent action, and (3) a *trust web* of signed, scoped, time-bounded, revocable, and delegatable capability grants with URI-based permission matching. We implement AgenticIdentity in 10,859 lines of Rust across four crates—core library, CLI, MCP server, and C FFI—with 182 tests and zero external service dependencies. Benchmarks on Apple M4 Pro show Ed25519 key generation in 8.80 µs, action signing in 11.55 µs, receipt verification in 21.77 µs, and trust chain verification (depth 2) in 43.51 µs—all sub-millisecond, enabling real-time use in interactive agent systems. AgenticIdentity is the fourth module in AgenticOS, providing the cryptographic foundation upon which agent contracts, communication, and orchestration are built.

1 Introduction

The transformer architecture [21] has produced language models capable of tool use [19], multi-step reasoning [24], and autonomous task execution [17]. Increasingly, these models operate as *agents*—software entities that observe their environment, make decisions, and take actions with real-world consequences: deploying code, sending emails, modifying databases, and managing infrastructure.

Yet every AI agent today is *anonymous*. When an agent takes an action, there is no cryptographic proof that *this specific agent* performed *this specific action*. Session identifiers vanish. API keys authenticate the human operator, not the agent. Logs can be tampered with. There is no mechanism for an agent to prove its identity to another agent, no way to grant an agent scoped permissions that can be revoked, and no portable identity that survives session boundaries or provider changes.

This creates four fundamental problems. **No accountability:** when something goes wrong, there is no chain of custody linking an action to an agent with mathematical certainty. **No trust delegation:** permissions are all-or-nothing API keys; there is no "trust this agent to read my calendar but not my email, and revoke if compromised." **No agent-to-agent authentication:** when Agent A receives a message from Agent B, there is no proof B sent it. **No portability:** an agent's identity is trapped in one platform; switching providers means starting from zero.

We present AgenticIdentity, a cryptographic system that addresses all four problems through three primitives:

1. **Identity Anchors** — Ed25519 key pairs where the public key *is* the agent's identity, with hierarchical key derivation (HKDF-SHA256) for scoped session, capability, and device keys, plus key rotation without identity loss.
2. **Action Receipts** — Signed proofs of every action, with hash-chain links for tamper-evident audit trails and optional witness co-signatures for multi-party non-repudiation.
3. **Trust Web** — Signed, scoped, time-bounded, revocable trust grants with URI-based capability matching, delegation chains with depth limits, and configurable revocation channels.

AgenticIdentity is the fourth module in the AgenticOS ecosystem [15, 14, 16], providing the cryptographic foundation upon which agent contracts, communication, and orchestration are built.

The remainder of this paper is organized as follows. Section 2 surveys existing identity systems and their limitations for AI agents. Section 3 presents the architecture in detail. Section 4 describes the Rust implementation. Section 5 reports micro-benchmarks and security analysis. Section 6 describes concrete use cases. Section 7 discusses related work. Section 8 concludes.

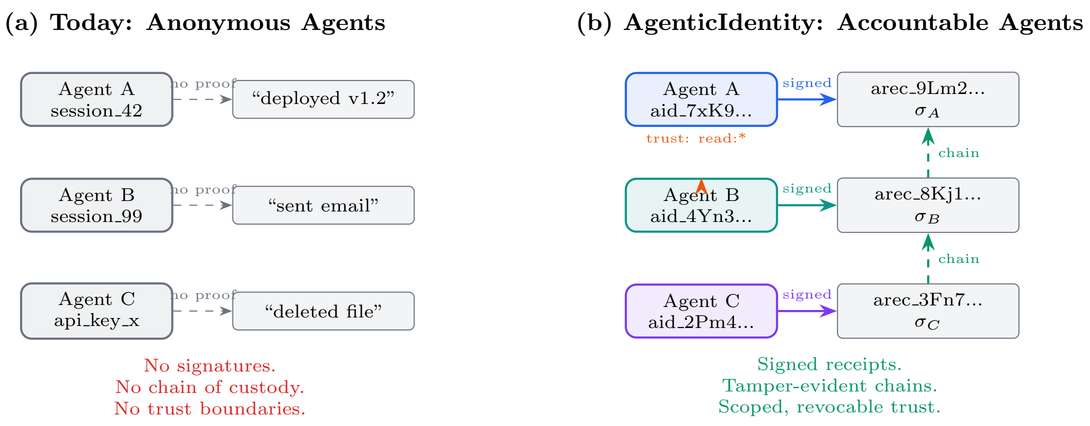

Figure 1: Before and after AgenticIdentity. (a) Today, agents are identified by ephemeral session tokens; actions produce no cryptographic proof. (b) With AgenticIdentity, each agent has a persistent Ed25519 identity anchor, every action produces a signed receipt linked into a hash chain, and trust relationships are explicit, scoped, and revocable.

2 Background

Existing identity systems were designed for humans, devices, or services—not for AI agents. We survey the primary approaches and explain why each falls short.

X.509 PKI [6] provides certificate-based identity with hierarchical trust through certificate authorities. However, the issuance process assumes a human or organizational entity, certificates are expensive to manage at the scale of ephemeral agent instances, and the rigid CA hierarchy is poorly suited to the peer-to-peer trust relationships that emerge in multi-agent systems.

Decentralized Identifiers (DIDs) [23] decouple identity from centralized authorities. While conceptually aligned with agent identity needs, DID methods typically require blockchain anchoring (adding latency and cost), the DID document lifecycle assumes human-driven key management, and the ecosystem lacks agent-specific capability semantics.

WebAuthn/FIDO2 [22] provides strong authentication for web users via hardware tokens and platform authenticators. The protocol is inherently interactive (challenge-response with user gestures) and assumes a browser context, making it unsuitable for headless agent processes.

SPIFFE/SPIRE [20] provides workload identity for cloud-native services. SPIFFE IDs (`spiffe://trust-domain/path`) identify services, not individual agents, and the system requires a SPIRE server for identity attestation. SPIFFE lacks action-level signing, receipt chains, and capability-scoped trust delegation.

Table 1 summarizes the gaps. AI agents need four properties that no existing system provides simultaneously: (1) *autonomous creation* without human ceremony, (2) *action-level signing* that binds every action to its actor, (3) *scoped trust delegation* with capability URIs and depth limits, and (4) *zero external dependencies*—no CA, no blockchain, no attestation server.

Table 1: Comparison of identity systems. AgenticIdentity is the only system providing all four properties required for AI agent identity.

System	Auto Create	Action Signing	Scoped Trust	Zero Deps
X.509 PKI	✗	✗	✗	✗
DIDs	✓	✗	✗	✗
WebAuthn	✗	✗	✗	✗
SPIFFE	✗	✗	✗	✗
Macaroons [4]	✓	✗	✓	✓
AgenticIdentity	✓	✓	✓	✓

Auto Create = no human ceremony; Action Signing = per-action cryptographic proof; Scoped Trust = capability-URI delegation with revocation; Zero Deps = no external service required.

3 Architecture

AgenticIdentity defines three interlocking cryptographic primitives, each addressing one dimension of the agent identity problem.

3.1 Identity Anchor

An identity anchor is an Ed25519 [3] key pair that serves as the permanent root of an agent's identity. The public key *is* the identity—mathematical, unforgeable, and permanent. The private key proves ownership and never leaves the agent's encrypted storage.

Identity ID. The agent's identifier is derived deterministically from the public key: $\text{id} = \texttt{aid\_}\|\text{base58}(\text{SHA-256}(\text{pk})[0:16])$. This 16-byte truncated hash provides 128 bits of collision resistance while keeping IDs compact and human-readable.

Key hierarchy. From the root signing key, scoped child keys are derived via HKDF-SHA256 [12] with structured context strings:

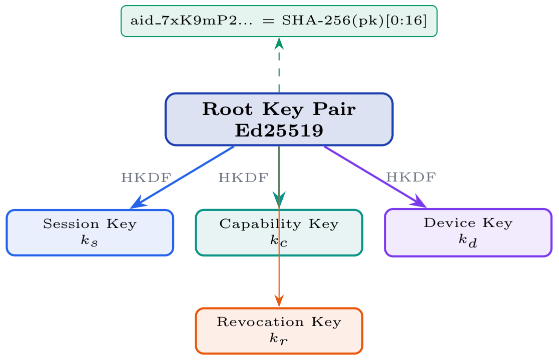

Figure 2: Identity anchor key hierarchy. The root Ed25519 key pair generates the permanent identity ID via SHA-256 truncation. Scoped child keys are derived via HKDF-SHA256 with structured context strings for session, capability, device, and revocation purposes.

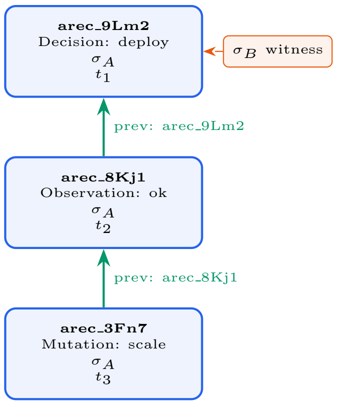

Figure 3: Receipt chain structure. Each receipt contains the actor's signature and optionally links to a previous receipt via its ID, creating a tamper-evident audit trail. Witness co-signatures enable multi-party attestation.

- **Session keys:** $k_s = \text{HKDF}(k_{\text{root}}, \texttt{"session/"} \| \text{sid})$ — ephemeral, auto-expire with the session.
- **Capability keys:** $k_c = \text{HKDF}(k_{\text{root}}, \texttt{"capability/"} \| \text{uri})$ — scoped to a specific permission.
- **Device keys:** $k_d = \text{HKDF}(k_{\text{root}}, \texttt{"device/"} \| \text{did})$ — bound to specific hardware.
- **Revocation keys:** $k_r = \text{HKDF}(k_{\text{root}}, \texttt{"revocation/"} \| \text{tid})$ — dedicated to revoking a trust grant.

This hierarchy enables the principle of least privilege: an agent can present a capability key proving it holds `read:calendar` authority without exposing its root identity key.

Key rotation. When a key is compromised or policy requires rotation, the agent generates a new key pair and signs a rotation record with the old key, authorizing the transition. The new identity document includes the full rotation history, enabling verifiers to trace the key lineage. Critically, the identity ID remains stable across rotations because it is derived from the *original* public key, not the current one—the agent's identity persists even as its keys evolve.

3.2 Action Receipts

An action receipt is a signed cryptographic object that proves an agent took a specific action at a specific time. Every receipt contains the actor's identity (public key), the action type (decision, observation, mutation, delegation, revocation, or custom), a structured content payload, a SHA-256 hash of the action content, and an Ed25519 signature over the hash.

Receipt IDs follow the same pattern as identity IDs: $\text{id} = \texttt{arec\_} \| \text{base58}(\text{SHA-256}(\text{receipt})[0\!:\!16])$.

Hash chaining. Each receipt can optionally reference a previous receipt by ID, creating a tamper-evident chain analogous to a blockchain but without consensus overhead. The receipt hash incorporates the previous receipt's ID, so any modification to an earlier receipt invalidates all subsequent hashes in the chain. An agent's action history is thus a verifiable, append-only log.

Witness signatures. Receipts support optional co-signatures from witness identities. When Agent A takes an action observed by Agent B, B can add its own signature to A's receipt, creating multi-party non-repudiation. Each witness signature includes the witness identity, timestamp, and Ed25519 signature over the receipt hash.

Verification. Receipt verification extracts the actor's public key from the receipt, recomputes the receipt hash from the content fields, and verifies the Ed25519 signature. For chained receipts, the verifier additionally checks that each link's previous-receipt reference is consistent.

3.3 Trust Web

A trust grant is a signed cryptographic object where Identity A declares: "I trust Identity B to perform {capabilities} under {constraints}." Trust grants are the mechanism by which agents receive authority and are held accountable for its use.

Capability URIs. Permissions are expressed as structured URI strings with colon-delimited hierarchies: `read:calendar`, `execute:deploy:production`, `storage/write`. Wildcard matching is supported: a grant of `read:*` covers any capability prefixed with `read:`. The universal wildcard `*` grants full authority.

Constraints. Each grant carries a `TrustConstraints` structure specifying: `not_before` (activation time), `not_after` (expiration), `max_uses` (count limit), and optional geographic or IP restrictions. Verification checks all constraints at query time.

Delegation. A grant can optionally allow the grantee to delegate authority to a third party. Delegation is controlled by two parameters: a boolean `delegation_allowed` flag and an integer `max_delegation_depth`. If Agent A grants to Agent B

Table 2: Cryptographic primitives used in AgenticIdentity.

Primitive	Algorithm	Purpose	Crate
Digital signature	Ed25519	Identity, signing	ed25519-dalek
Key exchange	X25519	Encrypted channels	x25519-dalek
Key derivation	HKDF-SHA256	Scoped child keys	hkdf
Password KDF	Argon2id	Encrypted storage	argon2
Sym. encryption	ChaCha20-Poly1305	.aid file encryption	chacha20poly1305
Hashing	SHA-256	Receipt/grant hashes	sha2
Memory safety	Zeroize	Private key clearing	zeroize

with depth 3, then B can delegate to C (depth 1), C to D (depth 2), and D to E (depth 3), but E cannot further delegate. Each delegated grant references its parent grant ID and increments the depth counter.

Revocation. Every trust grant includes a `RevocationConfig` specifying a revocation key ID, a revocation channel (local, HTTP endpoint, or distributed ledger), and optional required witnesses. Revocation produces a signed `Revocation` record published to the configured channel. Verification checks revocation status before accepting any grant.

Chain verification. For delegated trust, verification walks the entire chain from root to terminal grant, checking at each link: (1) signature validity, (2) time constraints, (3) revocation status, (4) capability coverage (each delegated grant must be a subset of its parent's capabilities), and (5) delegation depth within limits.

4 Implementation

AgenticIdentity is implemented in 10,859 lines of Rust organized as a Cargo workspace with four crates.

Core library (`agentic-identity`, 5,563 lines) provides the cryptographic primitives, data structures, storage layer, index engine, and query interface. Table 2 lists the cryptographic algorithms used.

The storage layer implements the `.aid` file format for encrypted identity persistence. Private keys are encrypted using a three-stage pipeline: the user's passphrase is processed through Argon2id [5] ($m = 65536$ KiB, $t = 3$ iterations, $p = 4$ lanes) to derive a master key, which is expanded via HKDF-SHA256 into an encryption key, then used with ChaCha20-Poly1305 [2] with a random 12-byte nonce. The public identity document is stored alongside in plaintext, enabling identity verification without decryption.

Receipt and trust stores are organized as directory trees of JSON files with in-memory indexes (HashMaps keyed by ID, actor, type; BTreeMaps for time-range queries). The query engine supports filtering by actor, type, time range, capability prefix, and validity status, with configurable sort order and result limits.

CLI (`agentic-identity-cli`, 1,121 lines) provides the `aid` binary with subcommands: `init`, `show`, `list`, `sign`, `verify`, `trust` (grant/revoke/list), `rotate`, `export`, and `receipt` (list). All commands support `--json` output for programmatic use.

Table 3: Performance benchmarks (Apple M4 Pro, release mode, Criterion 100 samples). All operations are sub-millisecond.

Operation	Latency
Ed25519 key generation	8.80 µs
Ed25519 sign	9.17 µs
Ed25519 verify	19.34 µs
HKDF key derivation	972 ns
Identity anchor creation	8.78 µs
Action receipt sign	11.55 µs
Action receipt verify	21.77 µs
Trust grant sign	12.41 µs
Trust grant verify	21.84 µs
Trust chain verify (depth 2)	43.51 µs
Receipt chain (10 receipts)	123.77 µs

MCP server (`agentic-identity-mcp`, 2,431 lines) implements the Model Context Protocol [1] over stdio with JSON-RPC 2.0 transport, exposing 10 tools (identity creation, action signing, receipt verification, trust management, health check) and 6 resources via `aid://` URI templates. The server enables any MCP-compatible LLM client—including Claude Desktop, VS Code with Copilot, and Cursor—to manage agent identity through natural language tool invocations.

FFI (`agentic-identity-ffi`, 1,176 lines) provides 11 C-compatible functions with opaque pointer handles and numeric error codes, enabling integration from Python, Node.js, Go, and any language with C FFI support.

Testing. The implementation includes 182 tests across all crates: 143 in the core library covering cryptographic operations, identity lifecycle, receipt creation and verification, trust grant semantics, capability matching, storage persistence, indexing, and query filtering; 30 in the MCP server testing all tools and resources; and 9 in the FFI crate testing the complete C API surface.

5 Evaluation

We evaluate AgenticIdentity along two dimensions: performance benchmarks to establish suitability for interactive agent systems, and security analysis to characterize the threat model and guarantees.

5.1 Performance Benchmarks

Hardware. Apple M4 Pro (ARM64), 48 GB unified memory, macOS 15.

Software. Rust 1.86.0, compiled with `--release` (full optimizations). All benchmarks use the Criterion [10] statistical framework with 100 iterations per measurement and outlier detection.

Table 3 reports the results. All operations complete in under 125 µs, well within the latency budget for interactive agent systems where tool calls typically take 100–500 ms of network round-trip time.

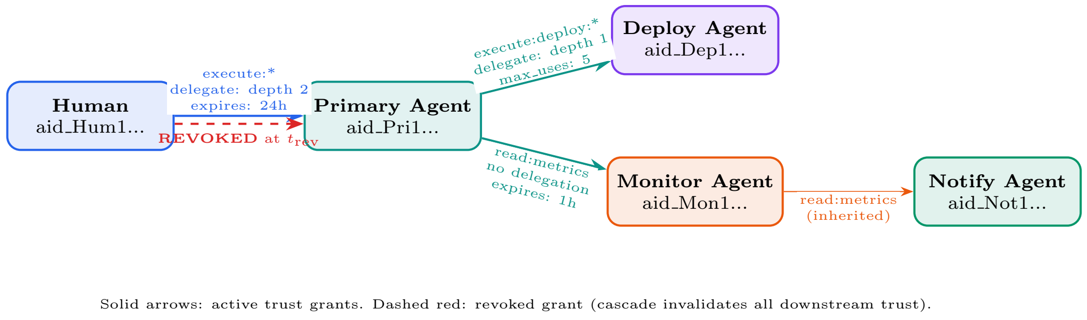

Figure 4: Trust web example showing delegation from a human operator through a hierarchy of agents. The human grants `execute:*` to the primary agent with delegation depth 2. The primary delegates scoped subsets to deploy and monitor agents. When the root grant is revoked (dashed red), all downstream trust is automatically invalidated during verification.

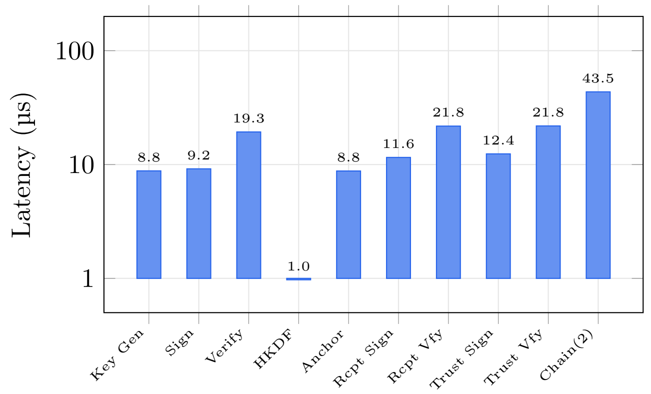

Figure 5: Benchmark results on log scale. All operations complete in under 50 μs except 10-receipt chain creation (124 μs, not shown). HKDF derivation at <1 μs enables on-the-fly key scoping.

Several observations merit discussion. Key generation at 8.80 μs is dominated by the Ed25519 scalar multiplication on Curve25519; this is >100× faster than the 1 ms target specified in our design requirements. Signing (9.17 μs) and verification (19.34 μs) exhibit the expected 2× asymmetry characteristic of Ed25519, where verification requires both scalar multiplication and point decompression.

Receipt signing (11.55 μs) adds only 2.38 μs over raw Ed25519 signing for SHA-256 hashing and JSON serialization of the action content. Trust grant verification (21.84 μs) is nearly identical to receipt verification, as both are dominated by the Ed25519 verify operation. Trust chain verification scales linearly with depth: at 43.51 μs for depth 2, we project depth 10 at ~218 μs, well under the 5 ms design target.

HKDF key derivation at 972 ns is noteworthy: deriving a scoped child key adds less than 1 μs, making on-the-fly key derivation practical for every tool invocation.

Table 4 translates these latencies into throughput figures for capacity planning.

Table 4: Throughput derived from benchmark latencies, assuming single-threaded operation.

Operation	**Throughput**
Identity creation	113,895/s
Action signing	86,580/s
Action verification	45,936/s
Trust grant creation	80,645/s
Trust verification	45,787/s
Key derivation (HKDF)	1,028,806/s

5.2 Security Analysis

Threat model. We assume the Dolev-Yao [8] network attacker model: the adversary can intercept, modify, replay, and inject messages on any network channel. Private keys are assumed secure (protected by Argon2id-encrypted storage with zeroization [18] on drop). We do not address side-channel attacks on the host platform.

Security properties. AgenticIdentity provides the following guarantees:

1. **Identity binding.** An identity ID is cryptographically bound to its public key via SHA-256 truncation. Forging an ID requires finding a SHA-256 preimage, which is computationally infeasible at 128 bits.
2. **Action non-repudiation.** A signed receipt proves the holder of the corresponding private key authorized the action. Under the assumption that private keys are not compromised, the actor cannot deny having taken the action.
3. **Chain integrity.** Tampering with any receipt in a chain invalidates all subsequent receipt hashes. The chain provides the same integrity guarantee as a hash chain (blockchain without consensus).
4. **Trust scoping.** Capability URI matching enforces that a grantee can only exercise permissions explicitly granted. Wildcard semantics are prefix-based: `read:*` matches `read:calendar` but not `write:calendar`.
5. **Revocation completeness.** Revoking a root grant in a delegation chain invalidates all downstream grants during verification, because chain verification checks every link.

Table 5: Security properties and their cryptographic foundations.

Property	Mechanism	Strength
Identity binding	SHA-256 → ID	128-bit preimage
Non-repudiation	Ed25519 signature	128-bit EUF-CMA
Chain integrity	SHA-256 hash chain	256-bit collision
Key secrecy	Argon2id + ChaCha20	256-bit symmetric
Key derivation	HKDF-SHA256	PRF security
Memory safety	Zeroize on drop	Prevents leak

6. **Key rotation continuity.** Key rotation preserves identity ID stability while the rotation record, signed by the old key, provides an authorization chain that verifiers can audit.

Table 5 summarizes the security properties and their cryptographic basis.

Limitations. AgenticIdentity does not address: (1) time authority—timestamps are taken from the host clock and are not externally attested; (2) key compromise recovery—if a root private key is extracted before rotation, the adversary can forge actions; (3) anonymous credentials—every action is attributable to an identity, which may conflict with privacy requirements. These represent directions for future work (Section 8).

6 Use Cases

Agent accountability. When a coding agent deploys a breaking change to production, the signed receipt chain proves exactly which agent took which actions in what order. Incident response becomes a matter of cryptographic verification rather than log archaeology.

Multi-agent coordination. In a system where multiple agents collaborate—a planner, a coder, a reviewer, and a deployer—each agent's identity anchor enables mutual authentication. The planner grants `execute:code:*` to the coder with a 1-hour expiry; the coder grants `read:diff` to the reviewer; the reviewer grants `execute:deploy:staging` to the deployer. Every delegation is scoped, time-bounded, and revocable.

Delegated authority with audit trails. A human operator grants an executive agent broad authority (`execute:*`, delegation depth 3). The executive delegates scoped subsets to specialist agents. When the human revokes the root grant—perhaps because the executive's behavior deviated from expectations—all downstream trust is automatically invalidated. The full receipt chain provides a complete audit trail for post-mortem analysis.

Cross-system portability. An agent's `.aid` file is a self-contained portable identity. The agent can move between Claude Desktop, VS Code, Cursor, or a custom MCP client without losing its identity, trust relationships, or action history. The MCP integration means any MCP-compatible tool can verify the agent's identity and check its authority before executing a request.

7 Related Work

SPIFFE/SPIRE [20] is the closest infrastructure-level analog, providing workload identity for cloud services via X.509-SVIDs or JWT-SVIDs. SPIFFE identifies *workloads*, not individual agent instances, and lacks per-action signing, receipt chains, and capability-scoped trust delegation. AgenticIdentity could complement SPIFFE by providing agent-level identity within a SPIFFE-authenticated service mesh.

Decentralized Identifiers [23] provide a standards-based framework for self-sovereign identity. While DIDs could serve as an alternative identity layer, most DID methods require blockchain anchoring (adding latency and infrastructure dependencies) and the DID Document specification does not define action receipts or trust delegation semantics. A future version of AgenticIdentity could expose identities as DID documents for interoperability.

Capability-based security [7, 13] is a long-standing approach to access control where a "capability" is an unforgeable token granting specific rights. Macaroons [4] extend this with contextual caveats that can attenuate authority. AgenticIdentity's trust grants are conceptually similar to attenuated macaroons with the addition of delegation chains, explicit revocation, and identity binding.

Blockchain identity systems [9, 11] provide decentralized, tamper-evident identity registries. However, they introduce consensus latency (seconds to minutes), require gas fees or permissioned networks, and do not address per-action signing or scoped trust for AI agents. AgenticIdentity achieves tamper-evidence through hash chains without consensus overhead.

AgenticOS ecosystem. AgenticIdentity is the fourth module in the AgenticOS suite. AgenticMemory [15] provides persistent cognitive memory via binary graph files. AgenticCodebase [14] provides semantic code navigation. AgenticVision [16] provides visual perception and persistent screenshots. AgenticIdentity provides the cryptographic trust layer that enables these modules to operate with accountability and verifiable authority.

8 Conclusion

We have presented AgenticIdentity, a cryptographic trust anchor that gives AI agents persistent, portable, mathematically verifiable identity. Through three interlocking primitives—identity anchors, action receipts, and a trust web—AgenticIdentity enables agent accountability, scoped trust delegation, tamper-evident audit trails, and cross-system portability, all without external service dependencies.

The implementation delivers strong performance in a compact footprint: 10,859 lines of Rust, 182 tests, four crates (core library, CLI, MCP server, FFI), and sub-50 µs latency for all individual cryptographic operations. Key generation at 8.80 µs, action signing at 11.55 µs, and trust verification at 21.84 µs confirm that cryptographic identity is practical for interactive agent systems where tool-call latency is measured in hundreds of milliseconds.

Future work. Three directions merit investigation. First, *verifiable timestamps* via an external timestamping authority (RFC 3161) or trusted execution environment would strengthen non-repudiation. Second, *selective disclosure*—enabling an agent to prove it holds a capability without revealing its full identity—would address privacy requirements in multi-tenant systems. Third, *DID interoperability* would allow AgenticIdentity anchors to participate in broader decentralized identity ecosystems.

Availability. AgenticIdentity is open-source under the MIT license. The source code, documentation, benchmarks, and this paper are available at `https://github.com/agentralabs/agentic-identity`.

References

[1] Anthropic. Model context protocol specification. `https://modelcontextprotocol.io/`, 2024. Accessed 2025.

[2] Daniel J Bernstein. ChaCha, a variant of Salsa20. Workshop Record of SASC 2008, 2008.

[3] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[4] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud, 2014.

[5] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[6] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and Tim Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, 2008.

[7] Jack B Dennis and Earl C Van Horn. Programming generality in a virtual machine environment. In *Proceedings of the Spring Joint Computer Conference*, pages 528–538. ACM, 1966.

[8] Danny Dolev and Andrew Yao. On the security of public key protocols. volume 29, pages 198–208. IEEE, 1983.

[9] Ethereum Foundation. ERC-725: Proxy account. `https://eips.ethereum.org/EIPS/eip-725`, 2017. Ethereum Improvement Proposal.

[10] Brook Heisler and Jorge Aparicio. Criterion.rs: Statistics-driven micro-benchmarking in Rust. `https://github.com/bheisler/criterion.rs`, 2023. Accessed 2025.

[11] Hyperledger Foundation. Hyperledger indy: Distributed ledger purpose-built for decentralized identity. 2019. Accessed 2025.

[12] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). *RFC 5869*, 2010.

[13] Mark S Miller. Robust composition: Towards a unified approach to access control and concurrency control. 2006.

[14] Omoshola Owolabi. AgenticCodebase: Semantic code graphs for AI agent navigation. 2026. Available at `https://github.com/agentralabs/agentic-codebase/tree/main/paper`.

[15] Omoshola Owolabi. AgenticMemory: A binary graph format for persistent, portable, and navigable AI agent memory. 2026. Available at `https://github.com/agentralabs/agentic-memory/tree/main/paper`.

[16] Omoshola Owolabi. AgenticVision: Visual cortex and persistent vision for AI agents. 2026. Available at `https://github.com/agentralabs/agentic-vision/tree/main/publication`.

[17] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

[18] RustCrypto Contributors. zeroize: Securely clear secrets from memory. `https://github.com/RustCrypto/utils/tree/master/zeroize`, 2024. Accessed 2025.

[19] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

[20] SPIFFE Project. SPIFFE: Secure production identity framework for everyone. `https://spiffe.io/`, 2024. CNCF Graduated Project. Accessed 2025.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[22] World Wide Web Consortium. Web authentication: An API for accessing public key credentials level 2. https://www.w3.org/TR/webauthn-2/, 2021. W3C Recommendation.

[23] World Wide Web Consortium. Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations. https://www.w3.org/TR/did-core/, 2022. W3C Recommendation.

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.