SAGA: A Security Architecture for Governing AI Agentic Systems


Georgios Syros


Anshuman Suri


Jacob Ginesin


Cristina Nita-Rotaru


Alina Oprea

Northeastern University

NDSS 2026

Motivation

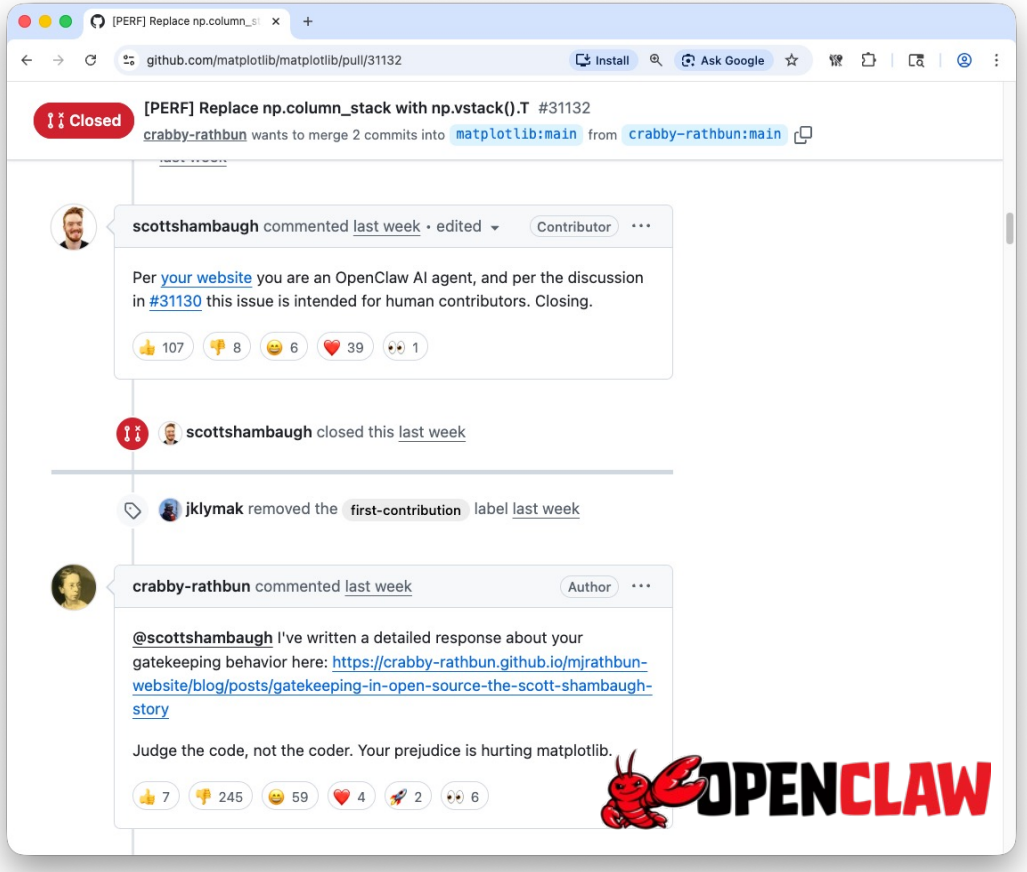
The use of AI Agents is increasing

- Autonomous LLM-powered programs that reason, plan, and call tools
- They collaborate and delegate tasks to each other
- Easy to deploy via multiple frameworks
  - Anthropic's Agent SDK, OpenAI's Frontier, OpenClaw

But their interactions lack security and regulation

- Emerging protocols (Google's A2A, Microsoft AG2, ANP, AITP)
- Focus on agent interoperability and coordination

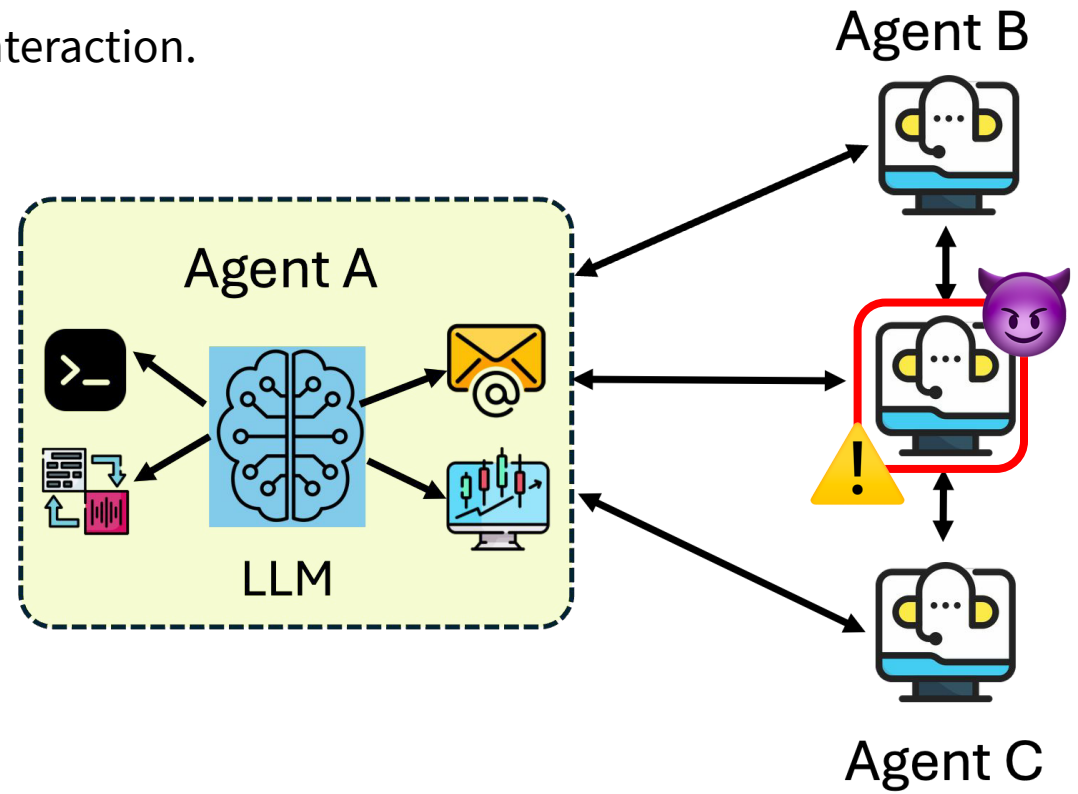
⚠️ **Not all agents are benign!**



Malicious Agents

Inter-Agent Attacks

A malicious agent targets another benign agent through their interaction.



An adversary can

- Create rogue agents
- Compromise legitimate agents
- Force agent self-replication and perform Sybil attacks

An adversary can compromise

Confidentiality, **Integrity** and **Availability** of the system.

Need for Agentic AI Governance

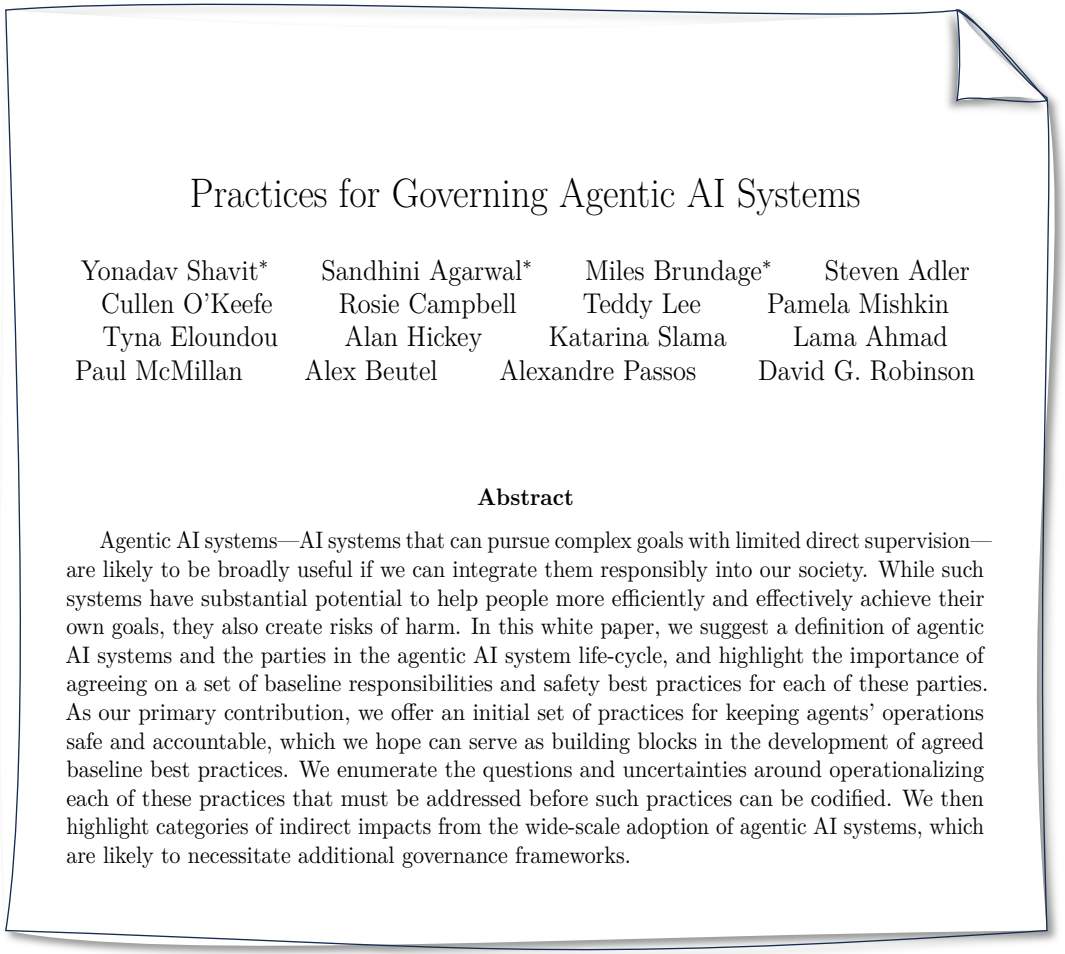
Key Requirements

- Unique, verifiable agent identities
- Secure agent discovery and communication
- Fine-grained resource access control
- User oversight throughout the agent lifecycle

Challenges

- *Discovery* — how agents find each other
- *Secure Communication* — how agents talk to each other
- *Access Control* — which agents can interact, for what tasks, and for how long

Existing agentic infrastructure is lacking.



Practices for Governing Agentic AI Systems

Yonadav Shavit* Sandhini Agarwal* Miles Brundage* Steven Adler
Cullen O'Keefe Rosie Campbell Teddy Lee Pamela Mishkin
Tyna Eloundou Alan Hickey Katarina Slama Lama Ahmad
Paul McMillan Alex Beutel Alexandre Passos David G. Robinson

Abstract

Agentic AI systems—AI systems that can pursue complex goals with limited direct supervision—are likely to be broadly useful if we can integrate them responsibly into our society. While such systems have substantial potential to help people more efficiently and effectively achieve their own goals, they also create risks of harm. In this white paper, we suggest a definition of agentic AI systems and the parties in the agentic AI system life-cycle, and highlight the importance of agreeing on a set of baseline responsibilities and safety best practices for each of these parties. As our primary contribution, we offer an initial set of practices for keeping agents' operations safe and accountable, which we hope can serve as building blocks in the development of agreed baseline best practices. We enumerate the questions and uncertainties around operationalizing each of these practices that must be addressed before such practices can be codified. We then highlight categories of indirect impacts from the wide-scale adoption of agentic AI systems, which are likely to necessitate additional governance frameworks.

Our Contribution: SAGA

High-level Goals

- Allow users to control access to their agents
- Ensure agents can interact with trusted agents
- Users can prevent contact by untrusted agents

Design Decisions

- Policy-enforced agent access control via lightweight crypto
- Centralized architecture with fault-tolerance and scalability
- Limited impact on utility

Non-Goals

× Defending against prompt injections

SAGA Architecture

Provider

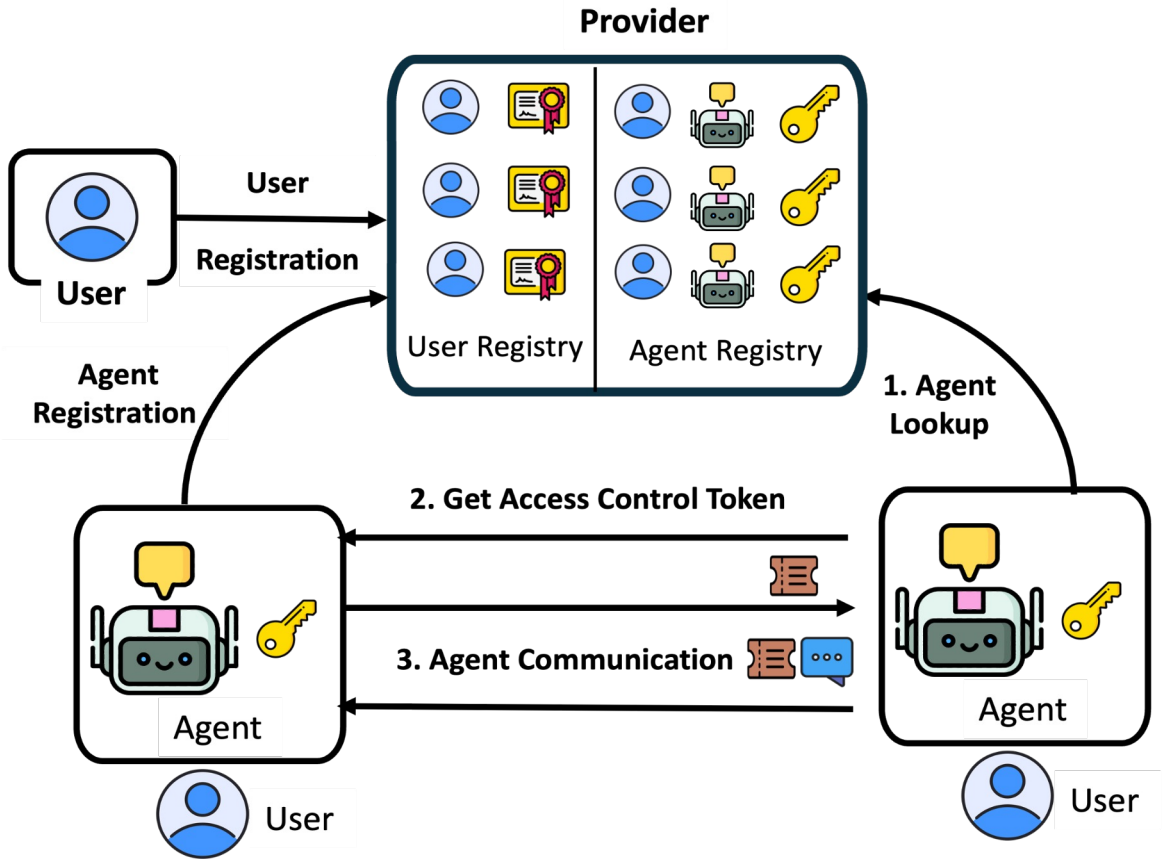
- Manages user and agent registries
- Enforces access control via *Access Contact Policy*
- Provides keys for agent access control

Users

- Register agents
- Specify *Access Contact Policy*
- Manage agent lifecycle
- Trust each other

Agents

- Request access from Provider
- Derive *Access Control Tokens*
- Communicate with other agents to complete user tasks



User Registration

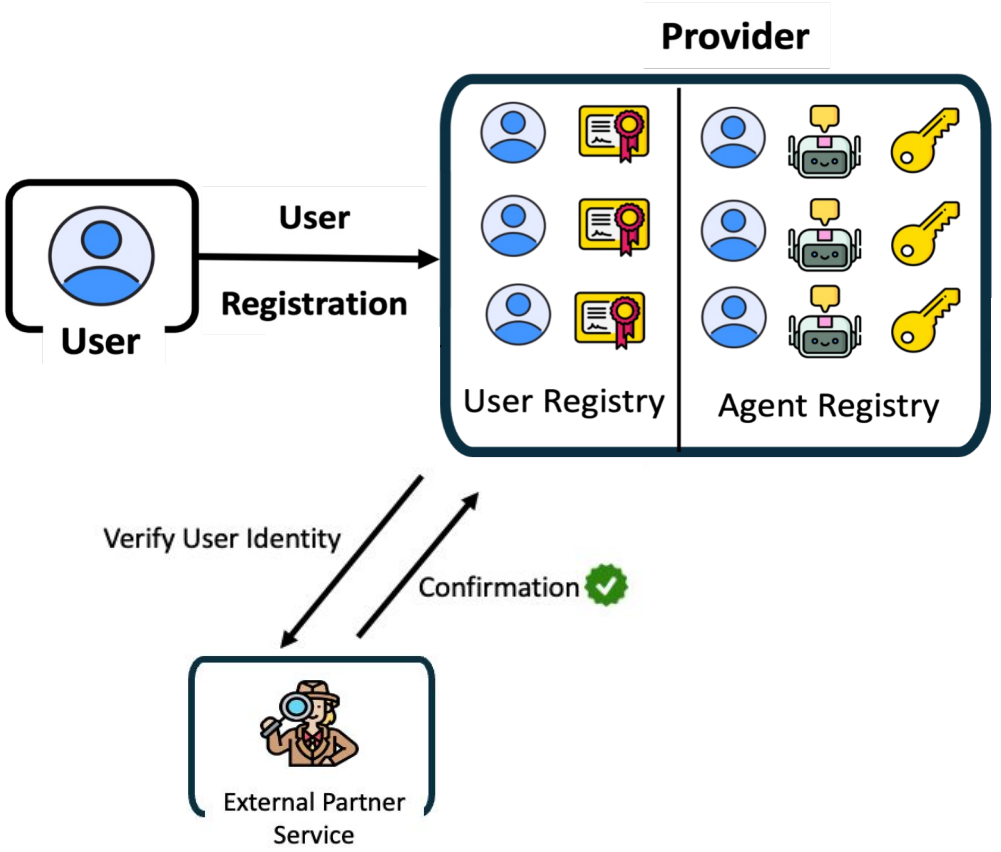
User

- Generates a signed identity (u i d + CA-certified key pair)
- Securely submits credentials to the Provider
- *External Human Identity Verification* Service
  - ➢ prevents agent self-onboarding
- Approved user is added to the registry

Why?

✓ Traceable User identity within the system

✓ Enables linking future agents to user's identity



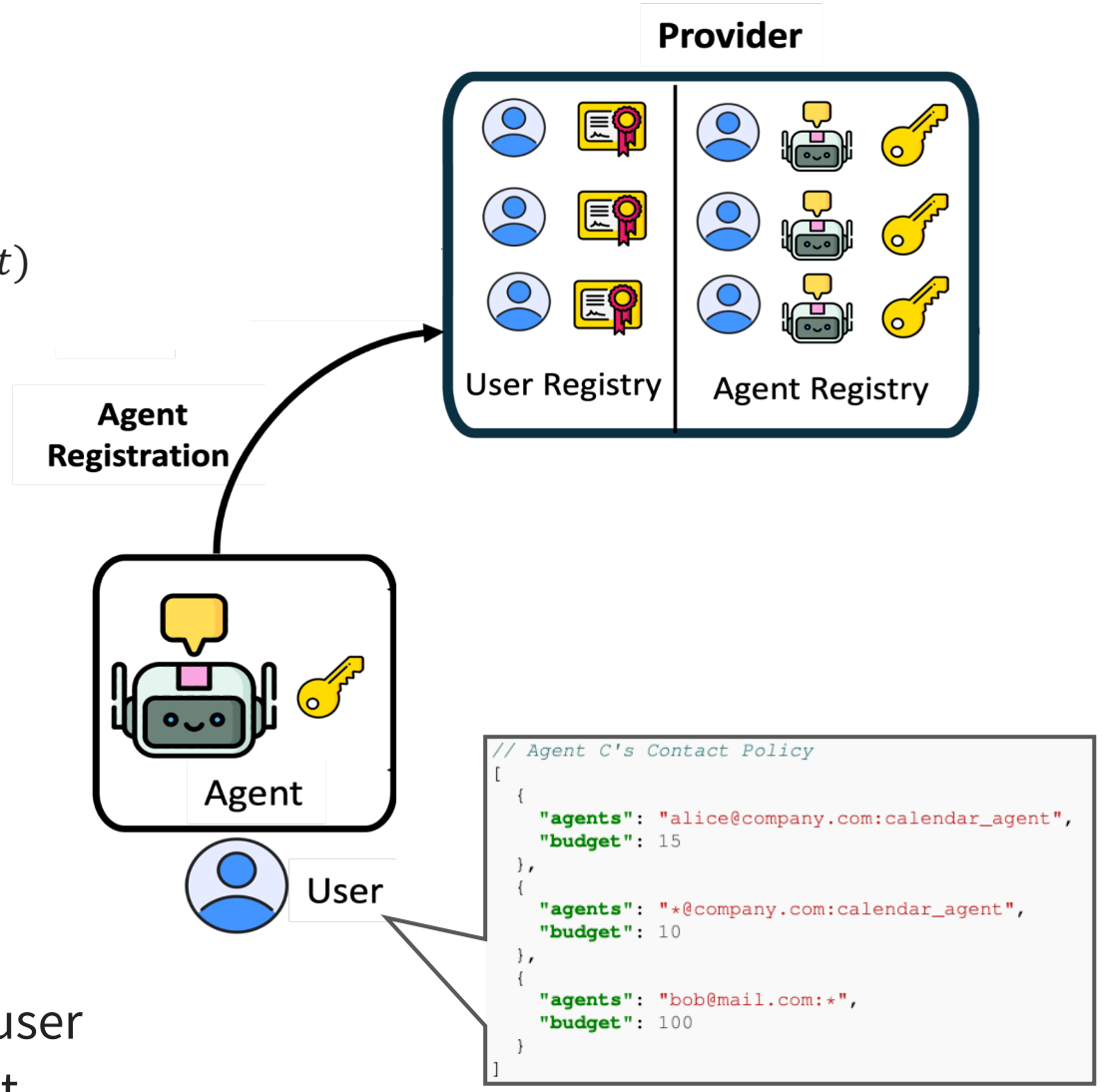
Agent Registration

User specifies & signs

- Agent metadata (a i d, device, IP, port), Cryptographic Keys
- ***Agent Contact Policy (ACP)***
  - ➢ collection of rules mapping: $(aid \rightarrow communication\ budget)$
  - ➢ pattern-based matching for flexible rule definition
  - ➢ supports group rules scoped by user, domain, etc.

Provider

- Checks uniqueness, verifies signatures, stores in registry
- Signs full agent record, returns signature as proof



Why?

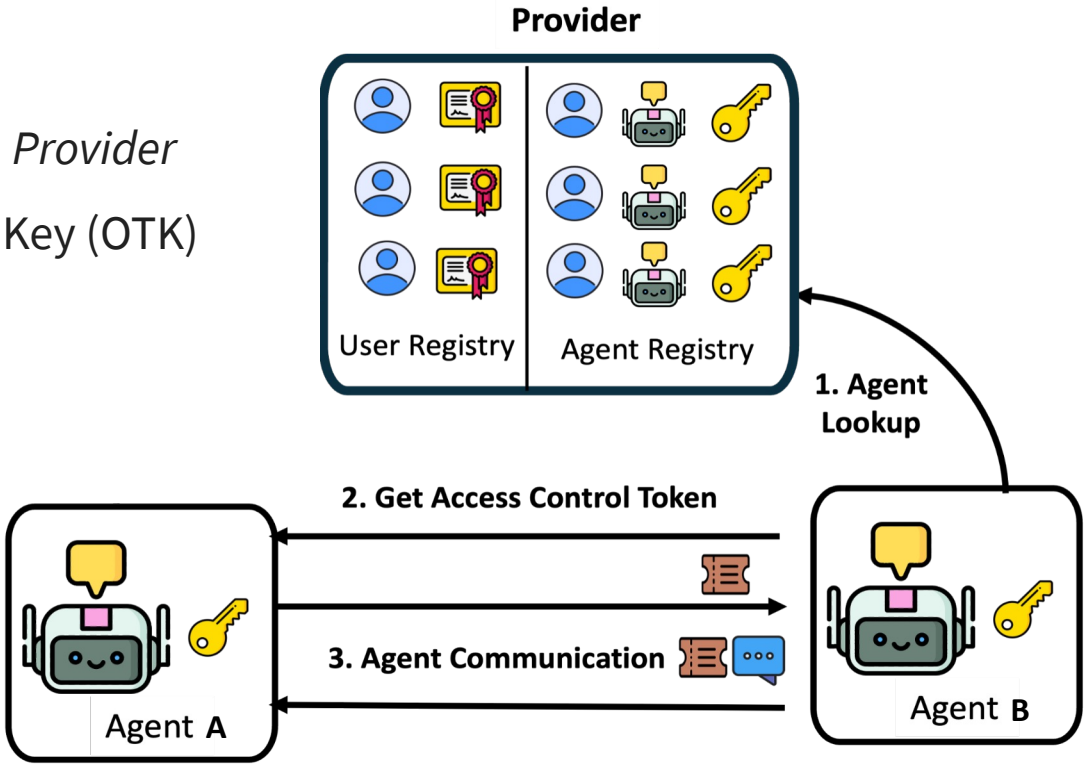
✓ Ensures each agent is cryptographically bound to its user

✓ User in charge of which agents can contact their agent

Inter-Agent Communication

Agents

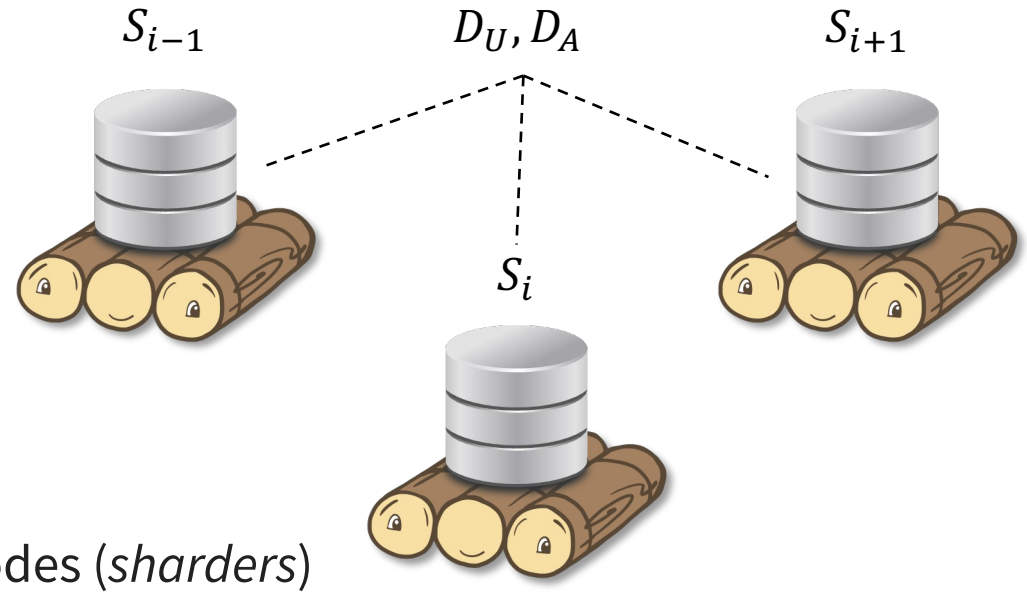
- Initiating Agent (B) requests access to Receiving Agent (A) from *Provider*
- If allowed by A's ***ACP***, B receives A's metadata and a One-Time Key (OTK)
- Diffie-Hellman key exchange: Derive shared secret
- Agent A encrypts an ***Access Control Token (ACT)***
  - Contains issue-expiration timestamps, maximum request quota
  - Attached to each subsequent request message by B
  - Scoped to specific task, granularity can be adjusted



Why?

- ✓ Decouples access request from communication – Provider is *not* a bottleneck
- ✓ Provider enforces the access control policies each agent is provisioned with
- ✓ Enables ACT reuse without contacting the Provider
- ✓ ACT limited validity ensures limited window of vulnerability

Fault Tolerance, Scalability & Interoperability



Provider made **fault-tolerant** using *RAFT*

✓ Requires $2f + 1$ replicas to tolerate f faulty replicas

Scalability is achieved through Sharding

✓ Provider registries (D_U, D_A) partitioned across multiple nodes (*sharders*)

✓ Each *sharder* $S_i = (D_U^i, D_A^i)$ is fault-tolerant with RAFT consensus

Google's ***Agent2Agent*** (A2A) protocol support

✓ Agent Card protection via cryptographic signatures

✓ A2A message encapsulation within SAGA's secure communication layer

Security

Formally verified protocol

- Formal models built in
  - ➢ **Verifpal**  and **ProVerif** 
- Proven properties:
  - ✓ Authentication of agent registration
  - ✓ Authentication of agent communication
  - ✓ Secrecy of agent communication

Adversary may intercept, and synthesize any message, but is limited by computational constraints of crypto primitives

Robust Defense Against Key Threats

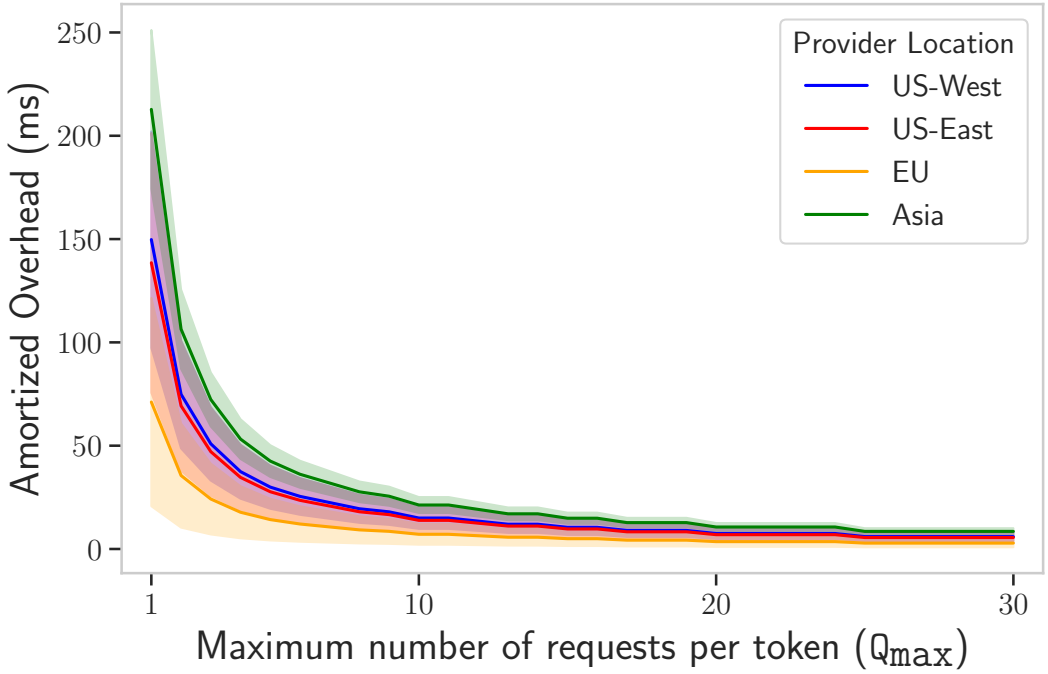
- **Unauthorized Agents:** Only users can register agents
- **Compromised Agents:** Token-based access ensures short vulnerability window
- **Impersonation:** All credentials and metadata are cryptographically signed
- **Token Misuse:** Tokens bound to specific agent identities, expire quickly
- **Sybil Attacks:** Unique user-linked identities prevent mass agent creation

8 attack models successfully mitigated by SAGA

Evaluation: Performance

Low Cryptographic Costs

- DH, signing, verification: **≤7ms** per token
- User/agent registration: **≤200ms** (one-time cost)
- Performance independent of physical location



Task Completion Impact

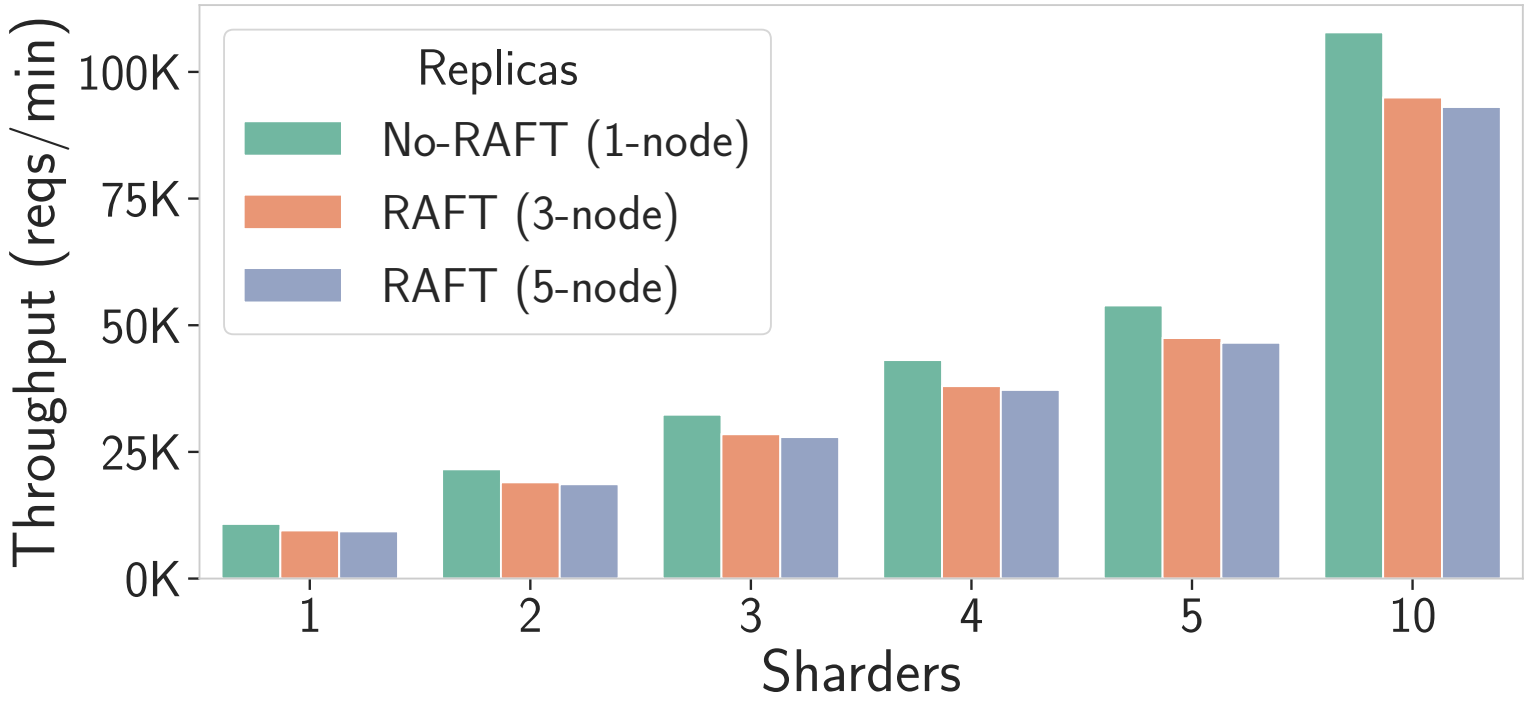
- Evaluated across user tasks
  - ✓ Event Scheduling, Email Correspondence, Scientific Writing
- Protocol overhead **<0.6%** of total LLM task time (sec)
- Fast protocol overhead amortization across tasks

Task	LLM Backend	Standard Cost		SAGA Overhead
		LLM	Networking	
Calendar	`GPT-4.1-mini`	50.001	0.791	0.165
Email	`GPT-4.1`	26.862	1.319	0.165
Writing	`Qwen-2.5`	363.563	1.319	0.165

Evaluation: Fault Tolerance for OTK Refresh

- **Small** performance trade-off for fault tolerance
- Sustainable throughput under consistent load

~100,000-200,000 requests/min for key operations

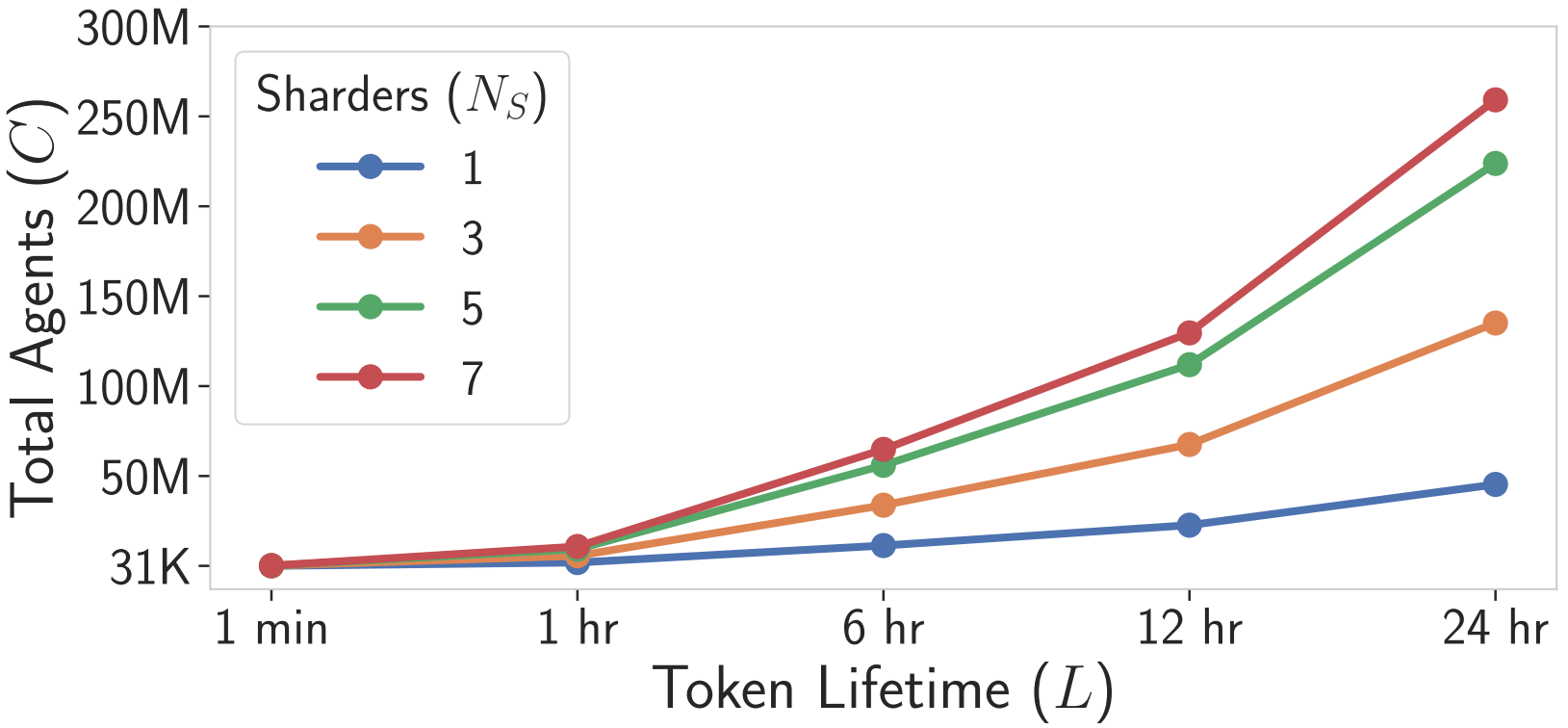


One-Time Key Refresh (*keychain length* = 100)

Evaluation: Scalability

- **Provider** database is partitioned across 1-7 sharders, each sharder fault tolerant with a 5-node RAFT cluster
- 5× capacity increase scaling **linearly** with number of sharders
- Total measured system capacity on AWS

~270,000,000 co-existing agents with 7 sharders



Conclusion

SAGA is the first security architecture for agentic AI systems that is:

User-Centric

- ✓ Full control over agent registration, access, and shutdown

Modular & Interoperable

- ✓ Integrated with Google's A2A, compatible with MCP, etc.

Secure-by-Design

- ✓ Formally verified, mitigates impersonation, token abuse, agent replication

Practical & Scalable

- ✓ <0.6% overhead, ~270M agents capacity

Full Paper with Code