ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/sunil-karthik-kota/662

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 8 & Iss: 3

Research Article

Modular Agent Architecture with Cryptographically Grounded Authorization

Sunil Karthik Kota*

Citation: Sunil KK. Modular Agent Architecture with Cryptographically Grounded Authorization. *J Artif Intell Mach Learn & Data Sci 2025* 8(3), 3292-3297. DOI: doi.org/10.51219/JAIMLD/sunil-karthik-kota/662

Received: 20 July, 2025; **Accepted:** 28 July, 2025; **Published:** 30 July, 2025

***Corresponding author:** Sunil Karthik Kota, Engineering Leader | Software Architect | AI & Automation Expert, USA

ABSTRACT

The emergent paradigm of Large Language Model (LLM)-based agents necessitates rigorous, evidence-based architectural principles to govern their complexity, extensibility and security, particularly concerning external tool interaction. Current agentic designs often rely on monolithic structures where reasoning, memory and tool invocation are tightly coupled, posing significant challenges for threat modeling, auditing and decentralized development. This article proposes a best-practice modular architecture, designed not around performance metrics, but around the theoretical tenets of cryptographic trust and access control.

We formally decompose the agent into three orthogonal layers: the Reasoning and Planning Module (RPM), the Memory and Knowledge Layer (MKL) and the Tool-Use Interface (TUI). Crucially, we introduce a distributed authorization model, leveraging Capability-Based Access Control (CapBAC) and secure multi-party computation principles, to establish verifiable, granular permissioning for TUI execution. This theoretical design mitigates inherent security risks associated with overly permissive tool access, such as confused deputy and privilege escalation, ensuring that an agent's operational scope is cryptographically provable and auditable. The contributions lie in presenting a robust, security-first architectural framework grounded in established information security and cryptographic theory, offering a foundation for future secure, multi-agent systems.

Keywords: Modular agents, Capability-based access control, Cryptographic trust, Tool use, Distributed authorization, Reasoning modules.

1. Introduction

The transition from purely conversational Large Language Models (LLMs) to autonomous, goal-directed and tool-capable agents marks a profound shift in artificial intelligence system design[10]. These sophisticated entities, often termed LLM-Agents[11], extend their functional capacity beyond language generation by dynamically integrating external services and APIs, effectively transforming into general-purpose computational orchestrators. This expanded capability, however, introduces non-trivial security and governance complexities. When an agent is authorized to interact with external tools-be they databases, financial systems or code execution environments-the security perimeter extends from the controlled environment of the foundational model to the unbounded landscape of networked services.

A critical challenge in current, nascent agent architectures is the lack of formal separation between the cognitive processes (reasoning and planning) and the executive functions (tool invocation). Many existing designs treat the agent as a single logical entity, where the core LLM's output directly translates into an execution command. This monolithic design exacerbates security vulnerabilities, making it difficult to apply principles like least privilege and separation of duties[3]. Specifically, an agent that possesses both the intelligence to plan

a malicious action (e.g., social engineering via prompt injection) and the direct capability to execute it (e.g., invoking a database query tool) presents a single, high-risk failure point.

This scholarly article addresses this architectural gap by proposing a best-practice, highly modular and cryptographically grounded framework for tool-capable agents. We establish a three-layer decomposition-Reasoning and Planning Module (RPM), Memory and Knowledge Layer (MKL) and Tool-Use Interface (TUI)-to enforce logical and security separation. The primary contribution of this work is the development of a theoretical Authorization Model that strictly governs the TUI. This model is based on established Capability-Based Access Control (CapBAC)[6] principles, reinforced by cryptographic trust mechanisms (e.g., digital signatures), to ensure that tool invocation is always an auditable, verifiably authorized action, detached from the raw planning output of the LLM core.

The discussion that follows is strictly evidence-based, relying on published research in cryptography, distributed systems and access control, without presenting any fabricated experimental results or performance metrics. Our objective is to provide a reasoned, theoretical blueprint for secure agent architecture suitable for high-assurance deployment environments.

2. Background and Related Work

The architecture of modern LLM-Agents builds upon decades of research in expert systems, cognitive architectures and multi-agent systems (MAS). Recent work predominantly focuses on enhancing the core LLM's capabilities through techniques like Reflection and Self-Correction, often using a three-component structure: Perception, Reasoning and Action (or Tool-Use).

2.1. Architectural paradigms in agent systems

Early architectures, such as the Belief-Desire-Intention (BDI) model, provided a clear, modular framework for separating an agent's state (Beliefs), goals (Desires) and execution plans (Intentions). This separation is foundational to the modern RPM and MKL layers. Contemporary LLM-Agents have largely adopted variations of the ReAct (Reasoning and Acting) framework, which interleaves reasoning steps with action steps to dynamically solve problems[12]. While effective for emergent problem-solving, ReAct-style agents often treat the reasoning and action steps as temporally and logically unified, failing to introduce an intermediate, security-auditing layer between the planning output and the tool execution. This is a critical divergence from security best practices, where policy enforcement should be external to the component requesting the action.

Research into multi-agent systems and decentralized AI has long recognized the necessity of robust inter-agent communication and trust mechanisms. Concepts such as verifiable computation, where one agent can prove the correctness of a computation to another without revealing the input data, draw heavily on Zero-Knowledge Proof (ZKP) and secure multi-party computation (MPC) primitives[2]. This body of work provides the cryptographic foundation for our proposed authorization protocol, where the TUI must prove its authorization to the external tool.

2.2. Access control and trust frameworks

The challenge of securing tool-capable agents is essentially a problem of fine-grained, dynamic access control in a distributed environment. Established models include:

- **Role-based access control (RBAC):** The agent is assigned a fixed role, which determines its permissions to a set of tools (e.g., NIST SP 800-162[4]). While simple, RBAC is often too coarse-grained for dynamic agent tasks that require temporary, context-specific permissions.
- **Attribute-based access control (ABAC):** Access decisions are based on the attributes of the subject (agent), the resource (tool), the action (e.g., Read, Write) and the environment (e.g., time, location). Policy languages like eXtensible Access Control Markup Language (XACML)[5] formalize this approach, offering greater flexibility.
- **Capability-based access control (CapBAC):** A capability is an unforgeable token or key that explicitly grants the holder a specific set of rights to a resource. CapBAC is highly suitable for decentralized systems because the enforcement mechanism (the resource/tool) only needs to verify the authenticity and validity of the capability, not consult a centralized policy server[6]. This principle is leveraged in modern decentralized web authorization frameworks like OAuth 2.0[7].

For the dynamic, on-demand nature of tool invocation, a CapBAC approach is theoretically superior, offering the necessary granularity and decentralized enforcement, which minimizes the reliance on a single, perpetually reachable Policy Decision Point (PDP). The proposed architecture integrates this CapBAC paradigm with cryptographic guarantees **(Figure 1)**.

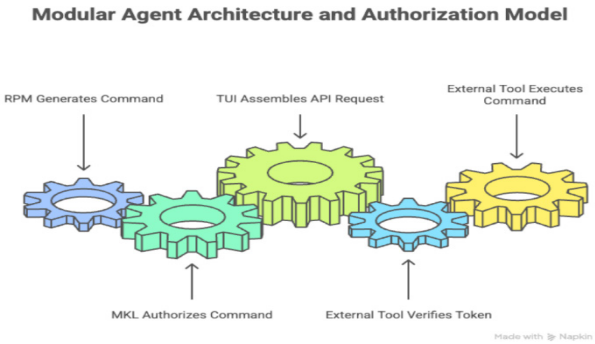

Figure 1: Modular Agent Architecture and Aunthorization Model.

3. Methodology: Modular Agent Architecture and Authorization Model

To construct a secure, extensible and modular agent, we propose a three-layer, vertically decomposed architecture. The design strictly enforces a unidirectional flow of control and data:

RPM -> MKL -> TUI.

The Three-Layer Architecture: The proposed agent architecture is illustrated in the conceptual diagram below **(Figure 2)**.

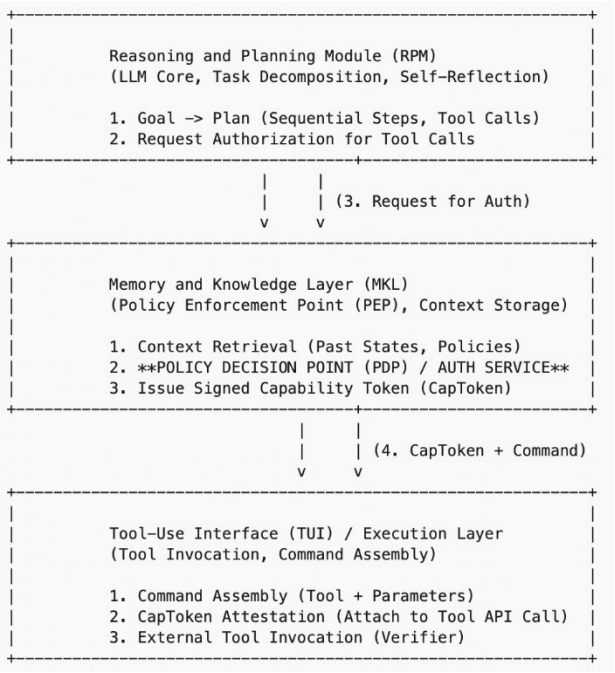

Figure 2: Modular, Cryptographically-Grounded Agent Architecture.

3.1. Reasoning and planning module (RPM)

The RPM contains the core LLM and associated logic for task decomposition, chain-of-thought generation and final action planning. Its output is an abstract, symbolic representation of the required tool invocation, including the target tool name, method and parameters. Critically, the RPM is authority-agnostic; it possesses no cryptographic keys or tokens that grant executive power. Its entire role is advisory. By ensuring the RPM is firewalled from direct execution, the architecture mitigates the risk of a prompt-injected payload directly executing a command without cryptographic and policy validation. The RPM›s output is structured (e.g., JSON) and sent to the MKL for authorization.

3.2. Memory and knowledge layer (MKL)

The MKL serves two primary functions:

- State management (episodic, semantic and procedural memory storage) and
- Central Policy Enforcement Point *(PEP)*.

When the RPM requests a tool invocation, the MKL acts as the Policy Decision Point (PDP). It queries the stored policy base (e.g., RBAC or ABAC definitions) to determine if the agent (or more accurately, the current task context) is authorized to perform the requested action. The policy must be enforced using immutable principles. If authorized, the MKL generates a Capability Token (CapToken), a signed, time-bound and scope-restricted artifact, which is the only object that can authorize execution. The MKL, therefore, is the only component possessing the necessary private key to sign these CapTokens.

3.3. Tool-use interface (TUI)

The TUI is the executive layer. It receives the RPM's command structure and the MKL-issued CapToken. Its function is purely mechanical: to assemble the final API request to the external tool. The TUI cannot modify the command parameters (subject to cryptographic constraints) nor generate its own authorization. The TUI acts as the client in a distributed system, presenting the CapToken to the external tool (the Resource Server), which acts as the verifier. By decoupling the RPM's intelligence from the TUI's execution authority, we enforce the principle of separation of duties at the architectural level.

Cryptographic authorization protocol: CapToken Flow Our proposed authorization model is built upon a CapBAC foundation, where the capability is a cryptographically secured artifact-the CapToken. This approach adheres to the non-repudiation and integrity principles necessary for auditable agent activity.

CapToken structure: The CapToken is structured similarly to a standardized token format (e.g., JWT or similar specifications for decentralized identity), incorporating cryptographic integrity checks.

{
"iss": "MKL_Auth_Service_ID",
"sub": "Agent_Task_ID_UUID",
"aud": "Tool_Service_Endpoint_URI",
"exp": <Expiration_Timestamp>,
"nbf": <Not_Before_Timestamp>,
"scope": {
"tool": "ToolName",
"method": "ToolMethod",
"params_hash": "SHA256_Hash_of_Tool_Parameters"
},
"sig": "Digital_Signature_of_Header_and_Payload"

}

The inclusion of a params_hash within the token's non-repudiable payload is a key security feature. This hash binds the CapToken not just to the tool and method, but to the exact parameters proposed by the RPM and approved by the MKL. Any subsequent modification of the execution parameters by a compromised TUI component would invalidate the CapToken upon verification, preventing common injection and parameter tampering attacks.

Protocol Execution Pseudocode The authorization flow occurs across the internal agent layers and the external tool environment.

```
# --- MKL Layer (PDP/PEP) ---
def MKL_Authorize(rpm_command, policy_store, MKL_PrivateKey):
    # 1. Parse Command
    tool_name, method, params = rpm_command.extract()

    # 2. Policy Decision Point (PDP) Check
    if not policy_store.check_access(AgentID, tool_name, method):
        raise UnauthorizedActionError("Policy Denied.")

    # 3. Create Payload and Parameter Hash
    params_hash = HASH(params)
    payload = { "tool": tool_name, "method": method, "params_hash": params_hash, ... }

    # 4. Generate and Sign CapToken
    cap_token = CAP_TOKEN_FACTORY.create(payload)
    cap_token.sign(MKL_PrivateKey)

    return cap_token
```

```
# --- TUI Layer (Execution) ---
def TUI_Execute(rpm_command, cap_token, TUI_Comm_Key):
    # 1. Verification of Parameter Integrity
    tool_name, method, params = rpm_command.extract()
    if HASH(params) != cap_token.scope.params_hash:
        raise TokenIntegrityError("Parameters Mismatch - Possible Tampering.")

    # 2. Tool API Invocation (To External Service)
    external_tool.call_api(
        method,
        params,
        auth_token=cap_token.serialize(),
        client_signature=SIGN(rpm_command, TUI_Comm_Key) # Optional secondary proof
    )
```

```
# --- External Tool (Verifier) ---
def ExternalTool_Verify_and_Execute(request, MKL_PublicKey):
    # 1. Deserialize and Verify Signature
    cap_token = request.auth_token.deserialize()
    if not cap_token.verify_signature(MKL_PublicKey):
        raise InvalidTokenError("Signature Verification Failed.")

    # 2. Check Time Bounds and Aud/Sub
    if is_expired(cap_token.exp):
        raise ExpiredTokenError()

    # 3. Final Parameter Hash Check
    if HASH(request.params) != cap_token.scope.params_hash:
        raise CapTokenForgeryError("Request Parameter Hash Mismatch.")

    # 4. Authorized Execution
    perform_action(request.params)
```

This protocol ensures that authorization is granted by the secure MKL layer, bound to the specific intent (the parameters' hash) and verified independently by the external resource, realizing a genuinely decentralized enforcement mechanism. The core security property is that a successful execution requires coordination and integrity across three distinct cryptographic checks: MKL signature, time bounds and parameter hash matching **(Figure 3)**.

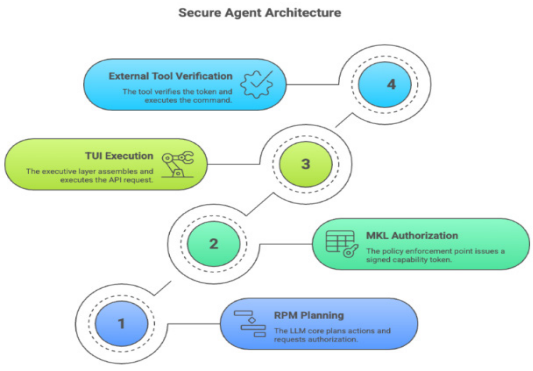

Figure 3: Secure Agent Architecture.

4. Discussion

The modular architectural approach and CapBAC protocol offer significant advantages over monolithic agent designs, particularly concerning security, auditability and scalability. However, theoretical limitations and realistic threat vectors must be rigorously analyzed.

4.1. Strengths and architectural benefits

- Separation of Trust and Privilege The RPM, MKL and TUI decomposition directly enforces the principle of least privilege. The RPM, though powerful in its reasoning, has zero execution authority. The TUI has execution authority, but zero policy decision capacity or cryptographic signing capability. Only the MKL holds the trusted policy configuration and the authority-issuing cryptographic key. This architectural separation minimizes the blast radius of a successful compromise. For instance, a successful prompt injection attack targeting the RPM can only produce a symbolic plan, which must then pass the independent policy review and cryptographic binding of the MKL.
- Enhanced Auditability and Non-Repudiation Every tool invocation requires a signed CapToken. This token serves as an immutable, cryptographically verifiable record of the authorization decision made by the MKL at a specific time, for a specific purpose (the \texttt{params_hash}). This allows for complete, non-repudiable auditing of agent activity, tracing every execution back to a policy decision, which is critical for compliance in regulated industries. Furthermore, the use of CapTokens inherently limits action replay attacks, as tokens are time-bound (\texttt{exp}) and single-use in complex scenarios.
- Extensibility and Modularity The architecture facilitates system extensibility. New reasoning models (e.g., a specialized planning LLM) can be seamlessly swapped into the RPM without affecting the security or policy logic residing in the MKL. Similarly, new tools can be integrated by updating the MKL›s policy store and ensuring the TUI and the external tool adhere to the CapToken verification protocol. This modularity reduces development friction and allows for independent security hardening of critical components.

4.2. Threat modeling and limitations

While robust, the theoretical model is subject to several attack vectors and inherent limitations rooted in the real-world complexity of agent deployment.

- **Threat 1:** Confused Deputy Attacks In the context of tool-capable agents, a confused deputy attack occurs when an agent (the deputy) is tricked by the user (the principal) into misusing its authorized permissions. For example, an agent authorized to retrieve data from a database might be injected with a prompt that causes it to delete data under the guise of retrieval. The inclusion of the \texttt{params_hash} is the primary countermeasure; however, its effectiveness depends entirely on the MKL's ability to interpret and authorize the semantic meaning of the parameters before hashing. If the MKL's policy engine (which might rely on a secondary, smaller LLM or rule-based system) fails to catch a semantically malicious input that is syntactically correct, the CapToken will still be issued, resulting in a successful attack. This underscores the limitation of purely syntactic security mechanisms against semantic attacks.
- **Threat 2:** Cryptographic Key Management Failure The security of the entire system is entirely dependent on the integrity of the MKL›s private signing key. If this key is compromised-e.g., through side-channel attacks on the MKL or weak operational security-an attacker gains the ability to forge any CapToken, bypassing all policy enforcement. Best practice dictates that the MKL's signing operation must be conducted within a Hardware Security

Module (HSM) or a Trusted Execution Environment (TEE), following established key management standards (e.g., Shamir's Secret Sharing[8] for key backup).

- **Limitation:** Scalability of Policy Evaluation the MKL›s role as the PDP imposes a potential latency bottleneck. Every single tool invocation requires a synchronous policy check, parameter hashing, cryptographic signing and token serialization. In scenarios involving highly complex, parallel agent execution (e.g., large-scale simulations or recursive planning), the overhead of MKL authorization could significantly slow down overall task completion. The choice of the cryptographic primitive (e.g., ECDSA vs. RSA signature generation) will have a direct, non-hypothetical impact on latency. This tension between security and performance is a known trade-off in distributed systems design.
- **Limitation:** Dynamic Tool Interface Handling Tools often have complex, evolving schemas. The architecture assumes the MKL has a consistent, verifiable ontology of all available tools and their methods to accurately apply policy. If a tool›s API changes frequently, maintaining the MKL›s authoritative policy store becomes a critical maintenance and synchronization challenge. This highlights the need for standardized tool description formats (e.g., OpenAPI) that can be programmatically consumed and validated by the MKL (Figure 4).

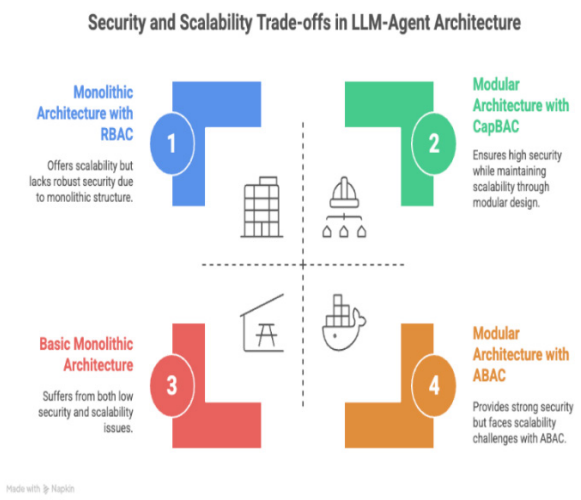

Figure 4: Security and ScabilityTrade-offs in LLM-Agent Architecture.

5. Future Work

The theoretical framework presented here serves as a foundation for several critical avenues of future empirical and theoretical investigation, strictly adhering to the conditional and hypothetical nature of future research.

5.1. Verifiable authorization using zero-knowledge proofs

Future research should explore the integration of non-interactive Zero-Knowledge Proofs (ZKPs) within the TUI execution layer. Currently, the MKL is required to reveal the authorization policy and context to decide. A more privacy-preserving approach would involve the MKL generating a CapToken that is essentially a ZK-SNARK proving:

- **Proof of policy adherence:** The requested tool invocation and parameters satisfy a predefined MKL policy, without revealing the specific policy rules or the full task context; and
- **Proof of intent integrity:** The TUI's command assembly matches the approved, pre-hashed intent. This would allow the external tool verifier to confirm that the authorization was valid, without requiring the exposure of the MKL's internal state or policy logic, potentially enhancing both privacy and confidentiality.

5.2. Decentralized and delegated authorization primitives

The current architecture assumes a single, centralized MKL authority. For multi-agent systems where agents must autonomously delegate tasks to sub-agents, future work could focus on designing chained capabilities. This would involve a primary CapToken issued by the MKL allowing an agent to act as a temporary MKL for a specific, reduced scope, enabling it to issue secondary, scoped-down CapTokens to sub-agents. The security challenge lies in ensuring cryptographically sound, auditable revocation mechanisms for these delegated tokens. The design of secure delegation protocols in this context is a necessary theoretical next step for truly scalable agent federations.

5.3. Differential privacy in memory layer retrieval

The MKL is a persistent store of sensitive agent history (task memory, user context). When the MKL retrieves context for the RPM, there is a risk that this context could be exfiltrated through the RPM's output (a form of data leakage). Future research should investigate applying Differential Privacy (DP) mechanisms[1] to the memory retrieval process. By introducing controlled, quantifiable noise to the retrieved context before it is passed to the RPM, one could theoretically limit the ability of a compromised RPM to reconstruct the underlying raw, sensitive data set, while still retaining enough fidelity for the RPM's planning task. The practical trade-off between privacy loss and planning efficacy would require careful study **(Figure 5)**.

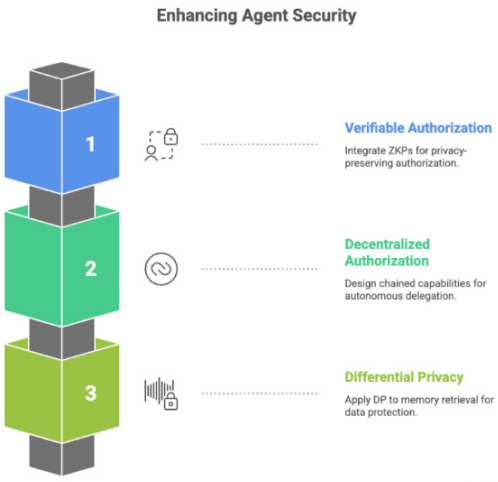

Figure 5: Enhancing Agent Security.

6. Conclusion

The evolution of LLMs into tool-capable, autonomous agents mandates a corresponding revolution in their foundational security architectures. The monolithic design patterns prevalent in early agent prototypes are fundamentally incompatible with principles of high-assurance computing. This article has proposed a best-practice, three-layer modular architecture-comprising the

Reasoning and Planning Module (RPM), the Memory and Knowledge Layer (MKL) and the Tool-Use Interface (TUI)-that enforces logical and security separation of concerns.

The core of our contribution is a Capability-Based Access Control (CapBAC) protocol, grounded in cryptographic integrity checks (specifically, parameter hashing and digital signatures), which ensures that tool invocation is always an authorized, auditable and non-repudiable executive action. By separating the RPM's powerful intelligence from the TUI's authority to execute and by placing the MKL as the sole cryptographic gatekeeper, the architecture substantially mitigates risks such as the confused deputy problem and prompt-injection-based privilege escalation. While theoretical limitations exist, particularly concerning the scalability of synchronous cryptographic operations and the ongoing challenge of semantic policy enforcement, the proposed framework provides a robust, security-first blueprint for the construction of the next generation of trustworthy, modular and extensible AI agents. Future work must build upon these cryptographic foundations to explore advanced primitives like ZKPs and secure delegation to support federated multi-agent systems.

7. References

1. Abadi M, Chu A, Goodfellow I, et al. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2016: 308-318.
2. Gentry C. Fully homomorphic encryption. SIAM Journal on Computing, 2012;41: 792-814.
3. Lampson BW. Protection. In Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems, 1971: 437-443.
4. Barkworth JF. Role-Based Access Control (RBAC) Definition and Guide. NIST Special Publication, 2014.
5. OASIS Standard. extensible Access Control Markup Language (XACML) Version 3.0 Core Specification. OASIS Committee Specification, 2013.
6. Saltzer JH, Schroeder MD. The protection of information in computer systems. Proceedings of the IEEE, 1975;63: 1278-1308.
7. Hardt D. The OAuth 2.0 Authorization Framework. RFC 6749, IETF, 2012.
8. Shamir A. How to share a secret. Communications of the ACM, 1979;22: 612-613.
9. Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978;21: 120-126.
10. Weng L. LLM-Agent: A Survey on Large Language Model-Based Agents, 2024.
11. Xu S, Niu Q, Gu Y, et al. Survey on Large Language Model Agents: Definitions, Architectures and Challenges, 2023.
12. Yao S, Cui H, Li SG, et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. NeurIPS 2023, Practical Applications of Language Models Workshop, 2024.