

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 1 October 2026

P. Kasselmann
Defakto Security
J. Lombardo
AWS
Y. Rosomakho
Zscaler
B. Campbell
Ping Identity
N. Steele
Open AI
30 March 2026

AI Agent Authentication and Authorization
draft-klrc-aiagent-auth-01

Abstract

This document proposes a model for authentication and authorization of AI agent interactions. It leverages existing standards such as the Workload Identity in Multi-System Environments (WIMSE) architecture and OAuth 2.0 family of specifications. Rather than defining new protocols, this document describes how existing and widely deployed standards can be applied or extended to establish agent authentication and authorization. By doing so, it aims to provide a framework within which to use existing standards, identify gaps and guide future standardization efforts for agent authentication and authorization.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://PieterKas.github.io/agent2agent-auth-framework/draft-klrc-aiagent-auth.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-klrc-aiagent-auth/>.

Source for this draft and an issue tracker can be found at <https://github.com/PieterKas/agent2agent-auth-framework>.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 1 October 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Conventions and Definitions	4
3. Agents are workloads	4
4. Agent Identity Management System	6
5. Agent Identifier	7
6. Agent Credentials	8
7. Agent Attestation	9
8. Agent Credential Provisioning	9
9. Agent Authentication	10
9.1. Transport Layer Authentication	10
9.1.1. Limitations	11
9.2. Application Layer Authentication	11
9.2.1. WIMSE Proof Tokens (WPTs)	11
9.2.2. HTTP Message Signatures	12
9.2.3. Limitations	12
10. Agent Authorization	13
10.1. Leverage OAuth 2.0 as a Delegation Authorization Framework	13
10.2. Use of OAuth 2.0 Access Tokens	13
10.3. Obtaining an OAuth 2.0 Access Token	14

- 10.3.1. User Delegates Authorization 14
- 10.3.2. Agent Obtains Own Authorization 14
- 10.3.3. Agents Accessed by Systems or Other Agents 15
- 10.3.4. OAuth 2.0 Security Best Practices 15
- 10.4. Risk Reduction with Transaction Tokens 15
- 10.5. Cross Domain Access 16
- 10.6. Human in the Loop 16
- 10.7. Tool-to-Service Access 17
- 10.8. Privacy Considerations {privacy-considerations} 18
- 10.9. OAuth 2.0 Discovery in Dynamic Environments 18
 - 10.9.1. Authorization Server Capability Discovery 18
 - 10.9.2. Protected Resource Capability Discovery 19
 - 10.9.3. Client Capability Discovery 19
- 11. Agent Monitoring, Observability and Remediation 19
- 12. Agent Authentication and Authorization Policy 21
- 13. Agent Compliance 21
- 14. Security Considerations 21
- 15. Privacy Considerations 21
- 16. IANA Considerations 21
- 17. Acknowledgments 21
- 18. Normative References 22
- Appendix A. Document History 25
- Authors' Addresses 26

1. Introduction

The rapid emergence of AI agents as autonomous workloads has sparked considerable innovation in authentication and authorization approaches. However, many of these efforts develop solutions in isolation, often reinventing existing mechanisms unaware of applicable prior art. This fragmentation risks creating incompatible implementations, duplicated development effort, and missed opportunities to leverage decades of established identity and authorization standards.

This document aims to help close that gap by providing a comprehensive model demonstrating how existing, well-established standards and some emergent specifications can be composed and applied to solve agent authentication and authorization challenges. Rather than proposing new protocols, this work focuses on integrating proven standards into a coherent framework tailored to the specific requirements of AI agent workloads.

By doing so, this document serves two complementary goals:

- 1. *Consolidation of prior art*: It establishes a baseline by showing how existing standards address the core identity, authentication, authorization, monitoring and observability needs

of agent-based systems. Implementers and standards developers can reference this framework to avoid redundant work and ensure interoperability.

2. *Foundation for future work*: As the agent ecosystem matures, having such a framework aids in identifying gaps and clarifies where extensions or profiles of existing standards are needed. This provides a foundation for more focused standardization efforts in areas needing novel work rather than variations of existing approaches.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Agents are workloads

An Agent is a workload that iteratively interacts with a Large Language Model (LLM) and a set of Tools, Services and Resources. An agent performs its operations until a terminating condition, determined either by the LLM or by the agent’s internal logic, is reached. It may receive input from a user, or act autonomously. Figure 1 shows a conceptual model of the AI Agent as a workload and illustrates the high-level interaction model between the User or System, the AI Agent, the Large Language Model (LLM), Tools, Services, and Resources.

In this document, Tools, Services, and Resources are treated as a single category of external endpoints that an agent invokes or interacts with to complete a task. Communication within or between Tools, Services, and Resources is out of scope.

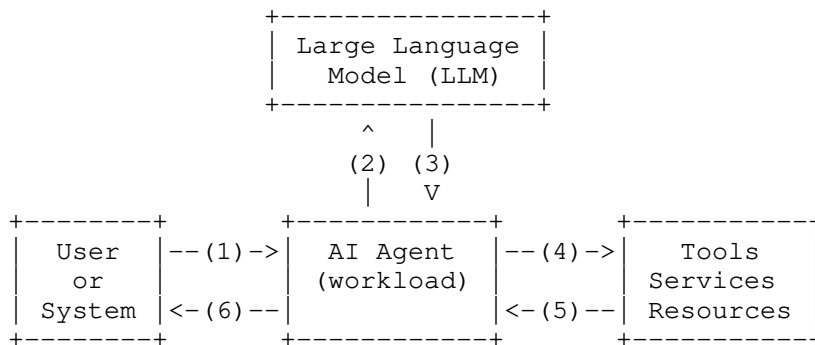


Figure 1: AI Agent as a Workload

1. Optional: The User or System (e.g. a batch job or another Agent) provides an initial request or instruction to the AI Agent.
2. The AI Agent provides the available context to the LLM. Context is implementation, and deployment, specific and may include User or System input, system prompts, Tool descriptions, prior Tool, Service and Resource outputs, and other relevant state.
3. The LLM returns output to the AI Agent facilitating selection of Tools, Services or Resources to invoke.
4. The AI Agent invokes one or more external endpoints of selected Tools, Services or Resources. A Tool endpoint may itself be implemented by another AI agent.
5. The external endpoint of the Tools, Services or Resources returns a result of the operation to the AI Agent, which may send the information as additional context to the Large Language Model, repeating steps 2-5 until the exit condition is reached and the task is completed.
6. Optional: Once the exit condition is reached in step 5, the AI Agent may return a response to the User or System. The AI Agent may also return intermediate results or request additional input.

As shown in Figure 1, the AI agent is a workload that needs an identifier and credentials so it can be authenticated by the Tools, Services, Resources, Large Language Model, System and the User (via the underlying operating system or platform, similar to existing applications and services). Once authenticated, these parties determine if the AI Agent is authorized to access the requested Large Language Model, Tools, Services or Resources. If the AI Agent is acting on behalf of a User or System, the User or System needs to delegate authority to the AI Agent, and the User or System context is preserved and used as input to authorization decisions and recorded in audit trails.

This document describes how AI Agents should leverage existing standards defined by SPIFFE [SPIFFE], WIMSE, OAuth and OpenID SSF [SSF].

4. Agent Identity Management System

This document defines the term Agent Identity Management System (AIMS) as a conceptual model describing the set of functions required to establish, maintain, and evaluate the identity and permissions of an agent workload. AIMS does not refer to a single product, protocol, or deployment architecture. AIMS may be implemented by one component or distributed across multiple systems (such as identity providers, attestation services, authorization servers, policy engines, and runtime enforcement points).

An Agent Identity Management System ensures that the right Agent has access to the right resources and tools at the right time for the right reason. An Agent identity management system depends on the following components to achieve its goals:

- * ***Agent Identifiers:** Unique identifier assigned to every Agent.
- * ***Agent Credentials:** Cryptographic binding between the Agent Identifier and attributes of the Agent.
- * ***Agent Attestation:** Mechanisms for determining and assigning the identifier and issue credentials based on measurements of the Agent's environment.
- * ***Agent Credential Provisioning:** The mechanism for provisioning credentials to the agent at runtime.
- * ***Agent Authentication:** Protocols and mechanisms used by the Agent to authenticate itself to Large Language Models or Tools (resource or server) in the system.
- * ***Agent Authorization:** Protocols and systems used to determine if an Agent is allowed to access a Large Language Model or Tool (resource or server).
- * ***Agent Observability and Remediation:** Protocols and mechanisms to dynamically modify the authorization decisions based on observed behavior and system state.
- * ***Agent Authentication and Authorization Policy:** The configuration and rules for each of the Agent Identity Management System.
- * ***Agent Compliance:** Measurement of the state and functioning of the system against the stated policies.

The components form a logical stack in which higher layers depend on guarantees provided by lower layers, as illustrated in Figure 2.

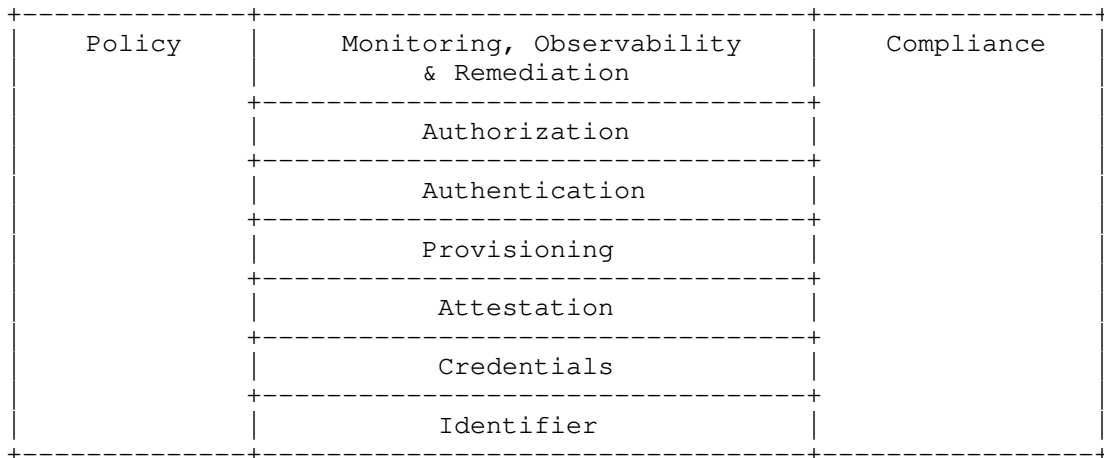


Figure 2: Agent Identity Management System

5. Agent Identifier

Agents MUST be uniquely identified in order to support authentication, authorization, auditing, and delegation.

The Workload Identity in Multi-System Environments (WIMSE) identifier as defined by [WIMSE-ID] is the primary identifier for agents in this framework.

A WIMSE identifier is a URI that uniquely identifies a workload within a trust domain. Authorization decisions, delegation semantics, and audit records rely on this identifier remaining stable for the lifetime of the workload identity.

The Secure Production Identity Framework for Everyone ([SPIFFE]) identifier is a widely deployed and operationally mature implementation of the WIMSE identifier model. A SPIFFE identifier ([SPIFFE-ID]) is a URI in the form of `spiffe://<trust-domain>/<path>` that uniquely identifies a workload within a trust domain.

An agent participating in this framework MUST be assigned exactly one WIMSE identifier, which MAY be a SPIFFE ID.

6. Agent Credentials

Agents **MUST** possess credentials that provide a cryptographic binding to the agent identifier. These credentials are considered primary credentials that are provisioned at runtime. An identifier alone is insufficient unless it can be verified to be controlled by the communicating agent through a cryptographic binding.

WIMSE credentials ([WIMSE-CRED]) are defined as a profile of X.509 certificates and Workload Identity Tokens (WITs), while SPIFFE defines SPIFFE Verified ID (SVID) profiles of JSON Web Token (JWT-SVID), X.509 certificates (X.509-SVID) and WIMSE Workload Identity Tokens (WIT-SVID). SPIFFE SVID credentials are compatible with WIMSE defined credentials. The choice of an appropriate format depends on the trust model and integration requirements.

Agent credentials **SHOULD** be short-lived to minimize the risk of credential theft, **MUST** include an explicit expiration time after which it is no longer accepted, and **MAY** carry additional attributes relevant to the agent (for example trust domain, attestation evidence, or workload metadata).

Deployments can improve the assurance of agent identity by protecting private keys using hardware-backed or isolated cryptographic storage such as TPMs, secure enclaves, or platform security modules when such capabilities are available. These mechanisms reduce key exfiltration risk but are not required for interoperability.

In some cases, agents **MAY** need secondary credentials to access a proprietary or legacy environment that is not compatible with the X.509, JWT or WIT it is provisioned with. In these cases an agent **MAY** exchange their primary credentials through a credential exchange mechanisms (e.g., OAuth 2.0 Token Exchange [OAUTH-TOKEN-EXCHANGE], Transaction Tokens [OAUTH-TXN-TOKENS] or Workload Identity Federation). This allows an agent to obtain a credential targeted to a specific environment by leveraging the primary credential in its possession.

Note: Static API keys are an antipattern for agent identity. They are bearer artifacts that are not cryptographically bound, do not convey identity, are typically long-lived and are operationally difficult to rotate, making them unsuitable for secure agent authentication or authorization.

7. Agent Attestation

Agent attestation is the identity-proofing mechanism for AI agents. Just as humans rely on identity proofing during account creation or credential issuance, agents require a means to demonstrate what they are, how they were instantiated, and under what conditions they are operating. Attestation evidence feeds into the credential issuance process and determines whether a credential is issued, the type of credential issued and the contents of the credential.

Multiple attestation mechanisms exist, and the appropriate choice is deployment and risk specific. These mechanisms may include hardware-based attestations (e.g., TEE evidence), software integrity measurements, supply-chain provenance, platform and orchestration-layer attestations, or operator assertions to name a few. Depending on the risk involved, a single attestation may be sufficient, or, in higher risk scenarios, multi-attestation may be required.

There are numerous systems that perform some form of attestation, any of which can contribute to establishing agent identity. For example, SPIFFE implementations can attest workloads using platform and environment specific mechanisms. At a high level, an attesting component gathers workload and execution context signals (such as where the workload is running and relevant platform identity attributes), presents those signals for verification to an issuer, and, as long as verification succeeds, binds the workload to a SPIFFE identifier and issues credentials (such as SVID) for subsequent authentication and authorization.

An agent identity management system may incorporate multiple attestation mechanisms and implementations to collect evidence and supply it to credential provisioning components. The selection of mechanisms depends on deployment constraints (such as the underlying platform and available identity signals) and the desired level of trust assurance.

8. Agent Credential Provisioning

Agent credential provisioning refers to the runtime issuance, renewal, lifecycle state and rotation of the credentials an agent uses to authenticate and authorize itself to other agents. Agents may be provisioned with one or more credential types as described in Section 6. Unlike static secrets, agent credentials are provisioned dynamically and are intentionally short-lived, eliminating the operational burden of manual expiration management and reducing the impact of credential compromise. Agent credential provisioning must operate autonomously, scale to high-churn environments, and integrate closely with the attestation mechanisms that establish trust in the

agent at each issuance or rotation event.

Agent credential provisioning typically includes two phases:

1. ***Initial Provisioning***: The process by which an agent first acquires a credential bound to its identity. This often occurs immediately after deployment or instantiation and is based on verified properties of the agent (e.g., deployment context, attestation evidence, or orchestration metadata).
2. ***Rotation/Renewal***: The automatic refresh of short-lived credentials before expiration. Continuous rotation ensures that credentials remain valid only for the minimum necessary time and that authorization state reflects current operational conditions.

The use of short-lived credentials provides a significant improvement in the risk profile and risk of credential exposure. It provides an alternative to explicit revocation mechanisms and simplifies lifecycle management in large, automated environments while removing the risks of downtime as a result of credential expiry.

Deployed frameworks such as [SPIFFE] provide proven mechanisms for automated, short-lived credential provisioning at runtime. In addition to issuing short-lived credentials, [SPIFFE] also provisions ephemeral cryptographic key material bound to each credential, further reducing the risks associated with compromising long-lived keys.

9. Agent Authentication

Agents may authenticate using a variety of mechanisms, depending on the credentials they possess, the protocols supported in the deployment environment, and the risk profile of the application. As described in the WIMSE Architecture [WIMSE-ARCH], authentication can occur at either the transport layer or the application layer, and many deployments rely on a combination of both.

9.1. Transport Layer Authentication

Transport-layer authentication establishes trust during the establishment of a secure transport channel. The most common mechanism used by agents is mutually-authenticated TLS (mTLS), in which both endpoints present X.509-based credentials and perform a bidirectional certificate exchange as part of the TLS negotiation. When paired with short-lived workload identities, such as those issued by SPIFFE or WIMSE, mTLS provides strong channel binding and cryptographic proof of control over the agent's private key.

mTLS is particularly well-suited for environments where transport-level protection, peer authentication, and ephemeral workload identity are jointly required. It also simplifies authorization decisions by enabling agents to associate application-layer requests with an authenticated transport identity. One example of this is the use of mTLS in service mesh architectures such as Istio or LinkerD.

9.1.1. Limitations

There are scenarios where transport-layer authentication is not desirable or cannot be relied upon. In architectures involving intermediaries, such as proxies, API gateways, service meshes, load balancers, or protocol translators, TLS sessions are often terminated and re-established, breaking the end-to-end continuity of transport-layer identity. Similarly, some deployment models (such as serverless platforms, multi-tenant edge environments, or cross-domain topologies) may obscure or abstract identity presented at the transport layer, making it difficult to bind application-layer actions to a credential presented at the transport layer. In these cases, application-layer authentication provides a more robust and portable mechanism for expressing agent identity and conveying attestation or policy-relevant attributes.

9.2. Application Layer Authentication

Application-layer authentication allows agents to authenticate independently of the underlying transport. This enables end-to-end identity preservation even when requests traverse proxies, load balancers, or protocol translation layers.

The WIMSE working group defines WIMSE Proof Tokens and HTTP Message Signatures as authentication mechanisms that may be used by agents.

9.2.1. WIMSE Proof Tokens (WPTs)

WIMSE Workload Proof Tokens (WPTs, [WIMSE-WPT]) are a protocol-independent, application-layer mechanism for proving possession of the private key associated with a Workload Identity Token (WIT). WPTs are generated by the agent, using the private key matching the public key in the WIT. A WPT is defined as a signed JSON Web Token (JWT) that binds an agent's authentication to a specific message context, for example, an HTTP request, thereby providing proof of possession rather than relying on bearer semantics.

WPTs are designed to work alongside WITs [WIMSE-CRED] and are typically short-lived to reduce the window for replay attacks. They carry claims such as audience (aud), expiration (exp), a unique token identifier (jti), and a hash of the associated WIT (wth). A WPT may

also include hashes of other related tokens (e.g., a Transaction Token) to bind the authentication contexts to specific transaction or authorizations details.

Although the draft currently defines a protocol binding for HTTP (via a Workload-Proof-Token header), the core format is protocol-agnostic, making it applicable to other protocols. Its JWT structure and claims model allow WPTs to be bound to different protocols and transports, including asynchronous or non-HTTP messaging systems such as Kafka and gRPC, or other future protocol bindings. This design enables receiving systems to verify identity, key possession, and message binding at the application layer even in environments where transport-layer identity (e.g., mutual TLS) is insufficient or unavailable.

9.2.2. HTTP Message Signatures

The WIMSE Workload-to-Workload Authentication with HTTP Signatures specification [WIMSE-HTTPSIG] defines an application-layer authentication profile built on the HTTP Message Signatures standard [HTTP-SIG]. It is one of the mechanisms WIMSE defines for authenticating workloads in HTTP-based interactions where transport-layer protections may be insufficient or unavailable. The protocol combines a workload's Workload Identity Token (WIT) (which binds the agent's identity to a public key) with HTTP Message Signatures (using the corresponding private key), thereby providing proof of possession and message integrity for individual HTTP requests and responses. This approach ensures end-to-end authentication and integrity even when traffic traverses intermediaries such as TLS proxies or load balancers that break transport-layer identity continuity. The profile mandates signing of some request components (e.g., method, request-target, content digest, and the WIT itself) and supports optional response signing. Note that @request-target covers only the request-target string (typically path + query) and not the method, scheme, or authority; those are only protected if separately covered (e.g., @method, @scheme, @authority).

9.2.3. Limitations

Unlike transport-layer authentication, application-layer authentication does not inherently provide channel binding to the underlying secure transport. As a result, implementations MUST consider the risk of message relay or replay if tokens or signed messages are accepted outside their intended context. Deployments typically mitigate these risks through short token lifetimes, audience restrictions, nonce or unique identifier checks, and binding authentication to specific requests or transaction parameters.

10. Agent Authorization

Agents act on behalf of a user, a system, or on their own behalf as shown in Figure 1 and need to obtain authorization when interacting with protected resources.

10.1. Leverage OAuth 2.0 as a Delegation Authorization Framework

The widely deployed OAuth 2.0 Authorization Framework [OAUTH-FRAMEWORK] is a mechanism for delegated authorization that enables an Agent to obtain limited access to a protected resource (e.g., a service or API), intermediated by an Authorization Server, often with the explicit approval of the authenticated User. An Agent uses OAuth 2.0-based mechanisms to obtain authorization from a User, a System, or on its own behalf. OAuth 2.0 defines a wide range of authorization grant flows that supports these scenarios. In these OAuth 2.0 flows, an Agent acts as an OAuth 2.0 Client to an OAuth 2.0 Authorization Server, which receives the request, evaluate the authorization policy and returns an access token, which the Agent presents to the Resource Server (i.e. the protected resources such as the LLM or Tools in Figure 1, which can evaluate its authorization policy and complete the request.

10.2. Use of OAuth 2.0 Access Tokens

An OAuth access token represents the authorization granted to the Agent. In many deployments, access tokens are structured as JSON Web Tokens (JWTs) [OAUTH-ACCESSTOKEN-JWT], which include claims such as 'client_id', 'sub', 'aud', 'scope', and other attributes relevant to authorization. The access token includes the Agent identity as the 'client_id' claim as defined in Section 2.2 of [OAUTH-ACCESSTOKEN-JWT].

When the Agent is acting on-behalf of another User or System, the User or System identifier is conveyed in the 'sub' claim as defined in Section 2.2 of [OAUTH-ACCESSTOKEN-JWT]. These identifiers MUST be used by resource servers protected by the OAuth 2.0 authorization service, along with other claims in the access token, to determine if access to a resource should be allowed. The access token typically includes additional claims to convey contextual, attestation-derived, or policy-related information that enables fine-grained access control. The resource server uses the access token and the information it contains along with other authorization systems (e.g. policy based, attribute based or role based authorization systems) when enforcing access control. JWT access tokens can be validated directly by resource servers while other formats that are opaque to the resource server can be validated through a mechanism that calls back to the authorization server (the mechanism is called

introspection despite the word having nearly the opposite meaning). This framework supports both models and does not require a specific token format, provided that equivalent authorization semantics are maintained.

A resource server in receipt of tokens opaque to it are able to obtain authorization and other information from the token through OAuth 2.0 Token Introspection [OAUTH-TOKEN-INTROSPECTION]. The introspection response provides the active state of the token and associated authorization attributes equivalent to those conveyed in structured tokens.

10.3. Obtaining an OAuth 2.0 Access Token

OAuth 2.0 defines a number authorization grant flows in support of different authorization scenarios. The appropriate flow depends on the specific authorization scenario and the nature of User involvement. The following subsections describe the most relevant flows for Agent authorization.

10.3.1. User Delegates Authorization

When a User grants authorization to an Agent for access to one or more resources (Tools, LLMs, etc.), the Authorization Code Grant, as described in Section 4.1 of [OAUTH-FRAMEWORK], is the canonical means of obtaining an access token. This redirection-based flow involves an interactive authorization process in which the user authenticates to the authorization server and explicitly approves the requested access. The resulting access token reflects the authorization delegated to the Agent by the User and can be used by the Agent to access resources on behalf of the user.

10.3.2. Agent Obtains Own Authorization

Agents obtaining access tokens on their own behalf can use the Client Credentials Grant as described in Section 4.4 of [OAUTH-FRAMEWORK] or the JWT Authorization Grant as described in Section 2.1 of [OAUTH-CLIENTAUTH-JWT]. When using the Client Credentials Grant, the Agent authenticates itself using one of the mechanisms described in Section 9 and not with the use of static, long-lived client secrets. When using the JWT Authorization Grant, the Agent will be identified in the subject of the JWT assertion.

10.3.3. Agents Accessed by Systems or Other Agents

Agents themselves can act in the role of an OAuth protected resource and be invoked by a System (e.g. a batch job) or another Agent. The System obtains an access token using an appropriate mechanism and then invokes the Agent presenting the access token.

10.3.4. OAuth 2.0 Security Best Practices

The Best Current Practice for OAuth 2.0 Security as described in [OAUTH-BCP] are applicable when requesting and using access tokens.

10.4. Risk Reduction with Transaction Tokens

Resources servers, whether they are LLMs, Tools or Agents (in the Agent-to-Agent case) may be composed of multiple microservices that are invoked to complete a request. The access tokens presented to the Agent, LLM or Tools can typically be used with multiple transactions and consequently have broader scope than needed to complete any specific transaction. Passing the access token from one microservice to another within an invoked Agent, LLM or the Tools increases the risk of token theft and replay attacks. For example, an attacker may discover and access token passed between microservices in a log file or crash dump, exfiltrate it, and use it to invoke a new transaction with different parameters (e.g. increase the transaction amount, or invoke an unrelated call as part of executing a lateral move).

To avoid passing access tokens between microservices, the Agent, LLM or Tools can exchange the received access token for a transaction token, as defined in the Transaction Token specification [OAUTH-TXN-TOKENS]. The transaction token allows for identity and authorization information to be passed along the internal call chain of microservices. The transaction token issuer enriches the transaction token with context of the caller that presented the access token (e.g. IP address, etc.), transaction context (transaction amount), identity information and a unique transaction identifier. This results in a downscoped token that is bound to a specific transaction and cannot be used as an access token, with another transaction, or within the same transaction with modified transaction details (e.g. change in transaction amount). Transaction tokens are typically short-lived, further limiting the risk in case they are obtained by an attacker by limiting the time window during which these tokens will be accepted.

A transaction token MAY be used to obtain an access token to call another service (e.g. another Agent, Tool or LLM) by using OAuth 2.0 Token Exchange as defined in [OAUTH-TOKEN-EXCHANGE].

10.5. Cross Domain Access

Agents often require access to resources that are protected by different OAuth 2.0 authorization servers. When the components in Figure 1 are protected by different logical authorization servers, an Agent SHOULD use OAuth Identity and Authorization Chaining Across Domains as defined in ([OAUTH-ID-CHAIN]), or a derived specification such as the Identity Assertion JWT Authorization Grant [OAUTH-JWT-ASSERTION], to obtain an access token from the relevant authorization servers.

When using OAuth Identity and Authorization Chaining Across Domains ([OAUTH-ID-CHAIN]), an Agent SHOULD use the access token or transaction token it received to obtain a JWT authorization grant as described in Section 2.3 of [OAUTH-ID-CHAIN] and then use the JWT authorization grant it receives to obtain an access token for the resource it is trying to access as defined in Section 2.4 of [OAUTH-ID-CHAIN].

When using the Identity Assertion JWT Authorization Grant [OAUTH-JWT-ASSERTION], the identity assertion (e.g. the OpenID Connect ID Token or SAML assertion) for the target end-user is used to obtain a JWT assertion as described in Section 4.3 of [OAUTH-JWT-ASSERTION], which is then used to obtain an access token as described in Section 4.4 of [OAUTH-JWT-ASSERTION].

OAuth Identity and Authorization Chaining Across Domains ([OAUTH-ID-CHAIN]) provides a general mechanism for obtaining cross-domain access that can be used whether an identity assertion like a SAML or OpenID Connect token is available or not. The Identity Assertion JWT Authorization Grant [OAUTH-JWT-ASSERTION] is optimized for cases where an identity assertion like a SAML or OpenID Connect token is available from an identity provider that is trusted by all the OAuth authorization servers as it removes the need for the user to re-authenticate. This is typically used within enterprise deployments to simplify authorization delegation for multiple software-as-a-service offerings.

10.6. Human in the Loop

An OAuth authorization server MAY conclude that the level of access requested by an Agent requires explicit user confirmation. In such cases the authorization server SHOULD either decline the request or obtain additional authorization from the User. An Agent, acting as an OAuth client, may use the OpenID Client Initiated Backchannel Authentication (CIBA) protocol. This triggers an out-of-band interaction allowing the user to approve or deny the requested operation without exposing credentials to the agent (for example a

push notification requesting the user to approve a request through an authenticator application on their mobile device).

Interactive agent frameworks may also solicit user confirmation directly during task execution (for example tool invocation approval or parameter confirmation). Such interactions do not by themselves constitute authorization and MUST be bound to a verifiable authorization grant issued by the authorization server. The agent SHOULD therefore translate user confirmation into an OAuth authorization event (e.g., step-up authorization via CIBA) before accessing protected resources.

This model aligns with user-solicitation patterns such as those described by the Model Context Protocol ([MCP]), where an agent pauses execution and requests user confirmation before performing sensitive actions. The final authorization decision remains with the authorization server, and the agent MUST NOT treat local UI confirmation alone as sufficient authorization.

**Note:* Additional specification or design work may be needed to define how out-of-band interactions with the User occur at different stages of execution. CIBA itself only accounts for client initiation, which doesn't map well to cases that envision the need for User confirmation to occur mid-execution.

10.7. Tool-to-Service Access

Tools expose interfaces to underlying services and resources. Access to the Tools can be controlled by OAuth and augmented by policy, attribute or role based authorization systems (amongst others). If the Tools are implemented as one or more microservices, it should use transaction tokens to reduce risk as described in Section 10.4 to avoid passing access tokens around within the Tool implementation.

Access from the Tools to the resources and services MAY be controlled through a variety of authorization mechanisms, including OAuth. If access is controlled through OAuth, the Tools can use OAuth 2.0 Token Exchange as defined in [OAUTH-TOKEN-EXCHANGE] to exchange the access token it received for a new access token to access the resource or service in question. When the Tool needs access to a resource protected by an authorization server other than the Tool's own authorization server, the OAuth Identity and Authorization Chaining Across Domains ([OAUTH-ID-CHAIN]) can be used to obtain an access token from the authorization server protecting that resource.

**Note:* It is an anti-pattern for Tools to forward access tokens it received from the Agent to Services or Resources. It increases the risk of credential theft and lateral attacks.

10.8. Privacy Considerations {privacy-considerations}

Authorization tokens may contain user identifiers, agent identifiers, audience restrictions, transaction details, and contextual attributes. Deployments SHOULD minimize disclosure of personally identifiable or sensitive information in tokens and prefer audience-restricted and short-lived tokens. Where possible, opaque tokens with introspection SHOULD be preferred when claim minimization is required.

Agents SHOULD request only the minimum scopes and authorization details necessary to complete a task. Resource servers SHOULD avoid logging full tokens and instead log token identifiers or hashes. When authorization context is propagated across services, derived or down-scoped tokens (such as transaction tokens) SHOULD be used to reduce correlation and replay risk.

Implementations MUST ensure that user identity information delegated to agents is not exposed to unrelated services and that cross-domain authorization exchanges only disclose information required for the target authorization decision.

10.9. OAuth 2.0 Discovery in Dynamic Environments

In dynamic Agent deployments (e.g., ephemeral workloads, multi-tenant services, and frequently changing endpoint topology), Agents and other participants MAY use OAuth discovery mechanisms to reduce static configuration and to bind runtime decisions to verifiable metadata.

10.9.1. Authorization Server Capability Discovery

An Agent that needs to obtain tokens can discover authorization server endpoints and capabilities using OAuth 2.0 Authorization Server Metadata [OAUTH-SERVER-METADATA] and/or OpenID Connect Discovery [OpenIDConnect.Discovery]. This allows the Agent to learn the authorization server issuer identifier, authorization and token endpoints, supported grant types, client authentication methods, signing keys (via `jwt_keys_uri`), and other relevant capabilities without preconfiguring them.

10.9.2. Protected Resource Capability Discovery

When an Agent is invoking a Tool, the Agent MAY use OAuth 2.0 Protected Resource Metadata [OAUTH-RESOURCE-METADATA] to discover how the resource is protected, including the resource identifier and the applicable Authorization Server(s) that protects Tool access. This enables an Agent to select the correct issuer/audience and token acquisition flow at runtime, even when resources are deployed or moved dynamically.

A Tool that attempts to access and OAuth protected resource MAY use OAuth 2.0 Protected Resource Metadata [OAUTH-RESOURCE-METADATA] in a similar way as an Agent. Similarly, a System may use [OAUTH-RESOURCE-METADATA] when accessing an Agent.

10.9.3. Client Capability Discovery

Other actors (e.g., Authorization Servers, registrars, or policy systems) may need to learn about any entities (System, Agent, Tool) that acts as OAuth clients. Where supported, they MAY use Client ID Metadata Documents [OAUTH-CLIENT-METADATA], which allow a client to host its metadata at a URL-valued `client_id` so that the relying party can retrieve client properties (e.g., redirect URIs, display information, and other registered client metadata) without prior bilateral registration.

As an alternative, entities acting as OAuth clients MAY register their capabilities with authorization servers as defined in the OAuth 2.0 Dynamic Client Registration Protocol [OAUTH-REGISTRATION].

11. Agent Monitoring, Observability and Remediation

Because agents may perform sensitive actions autonomously or on behalf of users, deployments MUST maintain sufficient monitoring and observability to reconstruct agent behavior and authorization context after execution. Observability is therefore a security control, not solely an operational feature.

Any participant in the system, including the Agent, Tool, System, LLM or other resources and service MAY subscribe to change notifications using eventing mechanisms such as the OpenID Shared Signals Framework [SSF] with either the Continuous Access Evaluation Profile [CAEP] or Risk Incident Sharing and Coordination [RISC] to receive security and authorization-relevant signals. Upon receipt of a relevant signal (e.g., session revoked, risk level change, subject disabled, token replay suspected, risk elevated), the recipient SHOULD remediate by attenuating access, such as terminating local sessions, discarding cached tokens, re-acquiring tokens with updated constraints, reducing

privileges, or re-running policy evaluation before continuing to allow access. Recipients of such signals MUST ensure that revoked or downgraded authorization is enforced without undue delay. Cached authorization decisions and tokens that are no longer valid MUST NOT continue to be used after a revocation or risk notification is received.

To support detection, investigation, and accountability, deployments MUST produce durable audit logs covering authorization decisions and subsequent remediations. Audit records MUST be tamper-evident and retained according to the security policy of the deployment.

At a minimum, audit events MUST record:

- * authenticated agent identifier
- * delegated subject (user or system), when present
- * resource or tool being accessed
- * action requested and authorization decision
- * timestamp and transaction or request correlation identifier
- * attestation or risk state influencing the decision
- * remediation or revocation events and their cause

Monitoring / Observability systems SHOULD correlate events across Agents, Tools, Services, Resources and LLMs to detect misuse patterns such as replay, confused deputy behavior, privilege escalation, or unexpected action sequences.

End-to-end audit is enabled when Agents, Users, Systems, LLMs, Tools, services and resources have stable, verifiable identifiers that allow auditors to trace "which entity did what, using which authorization context, and why access changed over time."

Implementations SHOULD provide operators the ability to reconstruct a complete execution chain of an agent task, including delegated authority, intermediate calls, and resulting actions across service boundaries.

12. Agent Authentication and Authorization Policy

The configuration and runtime parameters for Agent Identifiers Section 5, Agent Credentials Section 6, Agent Attestation Section 7, Agent Credential Provisioning Section 8, Agent Authentication Section 9, Agent Authorization Section 10 and Agent Monitoring, Observability and Remediation Section 11 collectively constitute the authentication and authorization policy within which the Agent operates.

Because these parameters are highly deployment and risk-model-specific (and often reflect local governance, regulatory, and operational constraints), the policy model and document format are out of scope for this framework and are not recommended as a target for standardization within this specification. Implementations MAY represent policy in any suitable `policy-as-code` or configuration format (e.g., JSON/YAML), provided it is versioned, reviewable, and supports consistent evaluation across the components participating in the end-to-end flow.

13. Agent Compliance

Compliance for Agent-based systems SHOULD be assessed by auditing observed behavior and recorded evidence (logs, signals, and authorization decisions) against the deployment's Agent Authentication and Authorization Policy Section 12. Since compliance criteria are specific to individual deployments, organizations, industries and jurisdictions, they are out of scope for this framework though implementers SHOULD ensure strong observability and accountable governance, subject to their specific business needs.

14. Security Considerations

TODO Security

15. Privacy Considerations

TODO Privacy but there's also Section 15...

16. IANA Considerations

This document has no IANA actions.

17. Acknowledgments

The authors would like to thank Sean O'Dell for providing valuable input and feedback on this work.

18. Normative References

- [A2A] "Agent2Agent (A2A) Protocol", n.d., <<https://github.com/a2aproject/A2A>>.
- [ACP] "Agentic Commerce Protocol", n.d., <<https://www.agenticcommerce.dev/docs>>.
- [AP2] "Agent Payments Protocol (AP2)", n.d., <<https://github.com/google-agentic-commerce/AP2>>.
- [CAEP] "OpenID Continuous Access Evaluation Profile 1.0", n.d., <https://openid.net/specs/openid-caep-1_0-final.html>.
- [HTTP-SIG] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <<https://www.rfc-editor.org/rfc/rfc9421>>.
- [MCP] "Model Context Protocol", n.d., <<https://modelcontextprotocol.io/specification>>.
- [OAUTH-ACCESSTOKEN-JWT]
Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <<https://www.rfc-editor.org/rfc/rfc9068>>.
- [OAUTH-BCP]
Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <<https://www.rfc-editor.org/rfc/rfc9700>>.
- [OAUTH-CLIENT-METADATA]
Parecki, A. and E. Smith, "OAuth Client ID Metadata Document", Work in Progress, Internet-Draft, draft-ietf-oauth-client-id-metadata-document-01, 1 March 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-client-id-metadata-document-01>>.
- [OAUTH-CLIENTAUTH-JWT]
Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <<https://www.rfc-editor.org/rfc/rfc7523>>.

[OAUTH-FRAMEWORK]

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/rfc/rfc6749>>.

[OAUTH-ID-CHAIN]

Schwenkschuster, A., Kasselmann, P., Burgin, K., Jenkins, M. J., and B. Campbell, "OAuth Identity and Authorization Chaining Across Domains", Work in Progress, Internet-Draft, draft-ietf-oauth-identity-chaining-08, 9 February 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-identity-chaining-08>>.

[OAUTH-JWT-ASSERTION]

Parecki, A., McGuinness, K., and B. Campbell, "Identity Assertion JWT Authorization Grant", Work in Progress, Internet-Draft, draft-ietf-oauth-identity-assertion-authz-grant-02, 2 March 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-identity-assertion-authz-grant-02>>.

[OAUTH-REGISTRATION]

Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/rfc/rfc7591>>.

[OAUTH-RESOURCE-METADATA]

Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <<https://www.rfc-editor.org/rfc/rfc9728>>.

[OAUTH-SERVER-METADATA]

Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <<https://www.rfc-editor.org/rfc/rfc8414>>.

[OAUTH-TOKEN-EXCHANGE]

Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/rfc/rfc8693>>.

- [OAUTH-TOKEN-INTROSPECTION]
Richer, J., Ed., "OAuth 2.0 Token Introspection",
RFC 7662, DOI 10.17487/RFC7662, October 2015,
<<https://www.rfc-editor.org/rfc/rfc7662>>.
- [OAUTH-TXN-TOKENS]
Tulshibagwale, A., Fletcher, G., and P. Kasselmann,
"Transaction Tokens", Work in Progress, Internet-Draft,
draft-ietf-oauth-transaction-tokens-08, 2 March 2026,
<<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-transaction-tokens-08>>.
- [OpenIDConnect.AuthZEN]
Gazitt, O., Ed., Brossard, D., Ed., and A. Tulshibagwale,
Ed., "Authorization API 1.0", 2026,
<https://openid.net/specs/authorization-api-1_0.html>.
- [OpenIDConnect.CIBA]
"OpenID Connect Client-Initiated Backchannel
Authentication Flow - Core 1.0", n.d.,
<https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html>.
- [OpenIDConnect.Discovery]
"OpenID Connect Discovery 1.0", n.d.,
<https://openid.net/specs/openid-connect-discovery-1_0-final.html>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [RISC] "OpenID Risk Incident Sharing and Coordination Profile
1.0", n.d.,
<https://openid.net/specs/openid-risc-1_0-final.html>.
- [SPIFFE] "Secure Production Identity Framework for Everyone", n.d.,
<<https://spiffe.io/docs/latest/spiffe-about/overview/>>.
- [SPIFFE-ID]
"SPIFFE-ID", n.d.,
<<https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE-ID.md>>.

[SSF] "OpenID Shared Signals Framework Specification 1.0", n.d.,
<https://openid.net/specs/openid-sharedsignals-framework-1_0-final.html>.

[WIMSE-ARCH] Salowey, J. A., Rosomakho, Y., and H. Tschofenig,
"Workload Identity in a Multi System Environment (WIMSE)
Architecture", Work in Progress, Internet-Draft, draft-
ietf-wimse-arch-07, 2 March 2026,
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-07>>.

[WIMSE-CRED] Campbell, B., Salowey, J. A., Schwenkschuster, A.,
Sheffer, Y., and Y. Rosomakho, "WIMSE Workload
Credentials", Work in Progress, Internet-Draft, draft-
ietf-wimse-workload-creds-00, 3 November 2025,
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-workload-creds-00>>.

[WIMSE-HTTPSIG] Salowey, J. A. and Y. Sheffer, "WIMSE Workload-to-Workload
Authentication with HTTP Signatures", Work in Progress,
Internet-Draft, draft-ietf-wimse-http-signature-02, 1
March 2026, <<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-http-signature-02>>.

[WIMSE-ID] Rosomakho, Y. and J. A. Salowey, "Workload Identifier",
Work in Progress, Internet-Draft, draft-ietf-wimse-
identifier-02, 2 March 2026,
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-identifier-02>>.

[WIMSE-WPT] Campbell, B. and A. Schwenkschuster, "WIMSE Workload Proof
Token", Work in Progress, Internet-Draft, draft-ietf-
wimse-wpt-01, 2 March 2026,
<<https://datatracker.ietf.org/doc/html/draft-ietf-wimse-wpt-01>>.

Appendix A. Document History

[[To be removed from the final specification]]

* latest

* Add Nick Steele from OpenAI as co-author.

-00

Authors' Addresses

Pieter Kasselmann
Defakto Security
Email: pieter@defakto.security

Jean-François Lombardo
AWS
Email: jeffsec@amazon.com

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com

Brian Campbell
Ping Identity
Email: bcampbell@pingidentity.com

Nick Steele
Open AI
Email: steele@openai.com