Open-Source Authenticated Delegation Framework – Technical Roadmap

Introduction

The rise of autonomous AI agents has created urgent challenges around **authorization and accountability**. When an AI agent acts on a user's behalf, third parties must be able to verify **(a)** that the actor is truly an AI agent, **(b)** which human user it represents, and **(c)** what permissions it has been grantedfile-tzfekfrsucjnj2cztgs6es. The proposed open-source library addresses these needs by implementing the framework from *"Authenticated Delegation and Authorized AI Agents"*. In essence, we will extend standard identity protocols (OAuth 2.0 and OpenID Connect) to support **agent-specific credentials and delegated authority**ar5iv.org. This library will enable human users to securely delegate tasks to Large Language Model (LLM) based agents (e.g. GPT or Claude), with fine-grained control over what the agent can do. Key features include issuing and verifying new token types for agents, translating natural-language instructions into formal access policies, and logging/auditing all agent actions. The end goal is a realistic **Minimum Viable Product (MVP)** ready for startup deployment – complete with cloud-ready architecture, APIs/SDKs, multi-tenant support, and a basic web UI for managing agents and permissions.

Objectives and Key Features

The project will deliver an open-source **"Authenticated Delegation"** library with the following capabilities:

- **Standards-Based Identity Integration:** Full compatibility with **OpenID Connect (OIDC)** and **OAuth 2.0** flows, so existing login systems and third-party OAuth providers (e.g. Google, Azure AD, Salesforce) can be leveraged. We will extend OIDC with new token types (described below) that carry agent identity and delegation infoar5iv.org. This ensures our approach works with established web authentication infrastructure.
- **Human-to-Agent Delegation:** Allow a human user to delegate a subset of their authority to an AI agent. The library will issue three kinds of tokens per the framework: a **User ID Token** for the human (standard OIDC ID Token), an **Agent ID Token** identifying the AI agent (as an OIDC "client credential"), and a **Delegation Token** binding the two and listing the agent's permissionsar5iv.orgar5iv.org. These tokens will be JWT-based (or similar) and digitally signed for verification.
- **Delegation Token Issuance & Verification:** Components to generate delegation tokens (e.g. as JWT with custom claims or W3C Verifiable Credentials). The **delegation token** will reference the user's ID token and agent's ID token (by unique ID or hash) and

include the allowed scope of actionsar5iv.org. It will also embed **validity conditions** (expiry, revocation URL) and be signed by the user (or their identity provider on behalf of the user) to prevent forgeryar5iv.org. Any third-party service or resource server that trusts the identity provider can verify the token, confirm the agent's authority, and check the scope before allowing accessar5iv.org.

- **Scoped Access Control & Policy Translation:** Tools for defining the delegation **scope** in natural language and converting it into structured access control rules. For example, a user might say *"Agent can read my emails and create calendar events, but not send emails"*, and the library will translate that into a formal policy (e.g. a JSON policy or XACML) that the agent's token will enforce. The framework from the paper outlines methods to express flexible natural-language permissions and transform them into auditable, fine-grained rulesfile-tzfekfrsucjnj2cztgs6es. We will leverage this by integrating an NLP component (potentially an LLM-based parser) to interpret permission descriptions and map them to policy configurations (role scopes, allowed operations, resource identifiers, etc.).

- **Human-in-the-Loop Controls:** Ensuring the human remains **in control**. The user can be prompted to review/approve high-risk actions or refine the agent's permissions if the natural-language instruction was ambiguous. The library will support optional **human confirmation steps** for certain agent requests (for example, an agent attempting a financial transaction might require the user's real-time approval). All agent actions can be recorded for later review.

- **Audit Logging and Accountability:** Every action the agent takes on a third-party service will be tied back to a delegation token and thus to the human user. The library will provide hooks to log each API call or transaction along with the responsible agent and user. This creates an audit trail for accountability. Service providers can use embedded metadata in the delegation token (e.g. an audit log URL or transaction ID) to report back or record what the agent didar5iv.org. Logs can be reviewed via the UI to detect misuse or for compliance.

- **Third-Party Service Integration:** Out-of-the-box support for common enterprise tools and APIs (Salesforce, Jira, Trello, Google Workspace, etc.) so that AI agents can safely interact with them. The library will include **integration adapters** or **SDK extensions** that handle translating the agent's delegation credentials into the target service's API calls. For example, if an agent is delegated permission to create tasks in Jira, the library could help exchange the delegation token for a Jira API token (using OAuth2 on-behalf-of flows) or inject the necessary OAuth token with restricted scope. We aim to provide example connectors for a few popular services to demonstrate how an agent can be securely integrated without sharing the user’s primary credentials.

- **Multi-Tenant Cloud Architecture:** Design the system to operate as a multi-tenant cloud service (e.g. an **Authorization-as-a-Service** for AI agents). This includes isolating data per tenant (organization or user group), supporting multiple identity providers (bring-your-own IdP or use a built-in one), and scaling to many users and agents. Multi-tenancy means a single deployment of the service could host delegation for many client applications or organizations, with proper segregation of credentials and policies. We’ll use cloud-native architecture (containerized microservices or a scalable monolith) that can be deployed on AWS/Azure/GCP or on-prem.
- **API and SDK for Developers:** A well-defined REST API (and possibly language-specific SDKs) will be provided for developers to integrate the delegation framework into their applications. For example, an app developer could call POST /delegate-token with a user’s and agent’s IDs and a policy to get back a signed delegation token. There will be endpoints for token introspection/verification, audit log retrieval, and policy management. SDKs in languages like Python or Node.js can wrap these APIs for easier use. We will ensure the API follows security best practices (OAuth 2.0 for the app to authenticate to our service itself, etc.) and provide clear documentation.
- **Basic Frontend UI:** A simple web interface where users can manage their AI agents and delegations. In the MVP, this might be a dashboard allowing a user to: register a new agent, view/edit the agent’s permitted actions (in both natural language and the generated policy form), revoke or rotate delegation tokens, and review audit logs of what the agent has done. This UI will make the framework more approachable and demonstrate its functionality (it’s also crucial if evolving into a startup product).

By achieving the above, the library will empower organizations and end-users to confidently deploy AI agents that can interact with online services in a controlled, auditable manner. Next, we detail the technical architecture and components needed to realize this.

High-Level Architecture

To implement these features, we propose a modular architecture consisting of an **Identity & Delegation Service**, a **Policy Engine**, and integration components, as shown below.

High-level architecture for authenticated delegation. The human user authenticates with an Identity Provider (OP) and registers an AI agent. The agent obtains an **Agent ID** and a **Delegation Token** (step 3 and 4 in the diagram) issued by the user/OP, allowing it to act on the user's behalf. The OP (acting as Authorization Server) and an optional UMA service coordinate to validate the agent's credentials (steps 1–4)ar5iv.org. When the agent calls a third-party **Resource Server** (RS) like an external API, it presents the delegation token and agent credentials. The RS can verify these via the OP, confirming the agent is authorized under the user's scope before granting access.

Identity & Delegation Service

At the core is an Identity Provider (IdP) extended to handle agent identities and delegation. We will build on an existing open-source OIDC server (such as **Keycloak**, **Gluu**, or **ORY Hydra**) to avoid reinventing user authentication. The IdP will be configured with:

- **User Authentication:** Standard OIDC user login (could be via social login, enterprise SSO, etc.). After the user logs in, they receive a normal OIDC **ID Token** representing their identity. This is the **User's ID-token**, which is "no different from those used in everyday login experiences"ar5iv.org.
- **Agent Registration:** The ability to register an AI agent as an OAuth2 client (likely a public client with no secret, since the agent might not have a secure storage for secrets). When an agent is registered, the IdP issues an **Agent-ID Token** for it – a signed token carrying information about the AI agent (unique agent ID, possibly its model or capabilities, etc.)ar5iv.org. The agent might authenticate to the IdP in the future by presenting this token or a corresponding credential. In practical terms, this could be done by treating the agent as a special type of user or client credential. The *Agent-ID token* extends a normal client credential with additional metadata (for example, JWT claims about the agent's software version, allowed tools, or a reference to a "system card" describing the agent).
- **Delegation Token Issuance:** An endpoint or protocol flow where a logged-in user can request issuance of a **Delegation Token** to a chosen agent. This could be implemented as an OAuth 2.0 extension (for instance, using the OAuth **JWT Authorization Grant** or a

custom grant type where the user (resource owner) authorizes a new token for the agent). Under the hood, the IdP or a companion service will create a signed token that includes: references to the User ID token and Agent ID token (e.g., their hashes or unique IDs) and the allowed scope of delegationar5iv.org. The token is digitally signed by the IdP **on behalf of the user** (with the user's consent), effectively meaning the **user delegates authority** to the agent via this token. The token also contains an expiration timestamp and may include a revocation URL or identifier such that it can be revoked if neededar5iv.org. Once issued, the Delegation Token is returned to the user (to give to the agent) or directly to the agent.

- **Token Introspection/Verification:** The service will expose an endpoint (per OAuth standards, e.g. introspect) that third-party services can call to verify a delegation token. However, to maintain compatibility, third parties might simply treat the delegation token as a JWT and verify its signature and claims locally (if they trust the IdP's keys). By including the user and agent references in the token, any service can check that the user and agent are valid and linked, and confirm the token's scope and validityar5iv.org. We will use standard JWT claims where possible (e.g., iss (issuer), sub (subject) for user, perhaps a custom claim for agent ID, and OAuth2 scope or a custom claim for permitted actions).

- **UMA Support:** The architecture will likely incorporate **User-Managed Access (UMA 2.0)**, which is an OAuth-based protocol for user-driven delegation. UMA fits well: the IdP can act as an UMA Authorization Server, where the user (resource owner) grants the AI agent (requesting party) a token to access certain resourcesar5iv.org. We can leverage UMA standards (Kantara UMA specification) to handle requests from the agent for access tokens to specific resources, governed by the user's policies. This adds a layer of standardized policy enforcement on top of raw tokens. In practice, this means if an AI agent wants to access a protected API (Resource Server), it would first obtain a token from the UMA server (with the delegation token as evidence of authorization). Keycloak, for instance, has built-in UMA support which we can adapt for agent delegation use cases.

Policy Engine for Scope Management

Defining and enforcing **scope limitations** on what an agent can do is one of the most challenging aspectsfile-tzfekfrsucjnj2cztgs6es. Our library will include a **Policy Engine** component to manage these constraints. Its responsibilities:

- **Natural Language Policy Translation:** When a user delegates to an agent, they should be able to specify permissions in human language. We will build a module (possibly using an LLM or a rule-based parser) that converts phrases into a structured policy. For example, “allowed to create and edit issues in Jira, but not delete” could be mapped to a JSON policy object:

json

CopyEdit

```
{ "actions": ["jira.issue.create", "jira.issue.edit"], "prohibited": ["jira.issue.delete"], "resources": ["project:XYZ"] }
```

We might leverage existing research or libraries for NL->policy translationfile-tzfekfrsucjnj2cztgs6es. An LLM like GPT-4 could be integrated (with a controlled prompt) to output a draft policy, which the user then **reviews in the UI** (human-in-the-loop) before it is attached to the delegation token. Over time, a library of common policy templates (for Salesforce, Jira, etc.) can be built to improve this process.

- **Policy Storage and Evaluation:** The formal policies will be stored (perhaps as part of the delegation token claims and/or in a database for easier querying). We will incorporate a policy evaluation engine such as **OPA (Open Policy Agent)** or an XACML library. When the agent attempts an action, the policy engine can check: does the policy allow this action on this resource? For on-the-fly decisions (like whether to auto-approve an action or require human approval), the policy engine will be consulted. For example, policy might say “allow agent to send emails *only if* the user has approved the draft” – such conditions would trigger a human-in-loop workflow.
- **Scope Enforcement in Tokens:** The delegation token itself will carry a **scope claim** or a pointer to the full policy. In many OAuth2 flows, scopes are simple strings (like read, write:calendar). We will design a scope vocabulary for typical AI agent actions (could be hierarchical like crm:read_contacts, crm:create_ticket, etc., or a URI referencing a policy document). Third-party services that understand our scheme could enforce scopes directly. For services that don’t, our integration layer (described below) will translate the scope into the service’s native permission (for instance, if the agent calls an API, the library could check scope before making the call).
- **Revocation and Refresh:** The policy engine will work with the Identity service to support **revocation** of delegation. If a user decides to revoke an agent’s access, the system can invalidate the delegation token (adding it to a revocation list or issuing a new generation that supersedes it). Similarly, policies might be set to **expire** after some time (short-lived tokens reduce risk). The library will provide a mechanism to refresh or renew delegation tokens, requiring user re-confirmation if needed, to maintain least-privilege.

Integration Adapters for Third-Party Services

A crucial part of this framework is making it easy to integrate AI agents into external services **without sacrificing security**. The library will offer an **Integration Adapter Layer** for popular services:

- **Credential Mediation:** Many third-party services (Salesforce, Google APIs, etc.) use OAuth 2.0 themselves. Our library can act as a broker: it can exchange the agent's delegation token for a service-specific access token with limited scope. One approach is the OAuth 2.0 *"On-Behalf-Of"* flow (supported in Azure AD and others) where the resource server accepts a token that asserts it is on behalf of a user+app. If possible, we will implement such flows, so that e.g. an agent with a delegation token for "Calendar access" can obtain a Google Calendar API token that is scoped appropriately. When direct exchange isn't possible, the adapter might need the user to pre-authorize the service (like connect their Salesforce account to our system). Then the library can store a refresh token and issue to the agent an access token for Salesforce constrained by the delegation policy.
- **API Connectors:** We plan to include sample connectors for a handful of services:
    - **Salesforce CRM:** An adapter that allows an agent to, say, create or update a Salesforce record on behalf of a user. The user would install a connected app in Salesforce that trusts our delegation service. The agent would present a delegation token to our adapter, which would then call Salesforce's REST API with an appropriate token or in an impersonation mode (Salesforce supports an OAuth JWT Bearer flow for impersonation). Only allowed CRM operations (per policy) will be executed.
    - **Atlassian Jira:** A connector to let an agent create Jira issues, comment, or move them through workflow. We might use Jira's OAuth and its fine-grained scopes (Jira has scopes for read/write on issues). The adapter will ensure the agent's actions (e.g. creating an issue) comply with the delegation (e.g. only in certain projects).
    - **Trello (or other project tools):** Similarly, allow creating or moving cards if permitted.
    - **Email/Calendar (Google or Outlook):** As a demo, an agent could manage a user's calendar. The library would store an OAuth token for Google Calendar with restricted access (perhaps using domain-wide delegation or the user's own token if they consent). The agent's requests go through a proxy that adds this token to call Google Calendar API, only if the delegation policy includes that action.

These adapters will illustrate how to integrate with real services; they will likely be implemented as plugins or separate modules using the library's core. In the MVP, we might implement one or two fully (e.g. a demo with an AI agent scheduling meetings via Google Calendar and creating tasks in Trello, under strict scope). This showcases end-to-end functionality.

Human-in-the-Loop and Audit Modules

To maximize user trust and safety, the system will incorporate **human oversight** and comprehensive logging:

- **User Approval Workflow:** The library can be configured so that certain agent actions require explicit user approval. For instance, an agent composing an email might need the user to click "Send" in the UI (unless policy says the agent can send on its own). We will include a lightweight workflow engine – perhaps something as simple as flagging the action and exposing an API/notification for the user to approve, which when approved triggers the action to resume. This could be integrated via the frontend (e.g. a notification in the dashboard or an email/SMS to the user for urgent tasks).
- **Real-Time Monitoring:** An admin or user could observe what an agent is doing in real-time. The system could provide a streaming log or feed of agent actions. For example, every time the agent uses a token to call a service, it's logged ("Agent X accessed *POST /api/contacts* on Salesforce at 10:30, allowed by policy Y"). This transparency builds trust that the agent isn't sneaking off-scope.
- **Audit Logs and Reporting:** All significant events (token issuance, token usage, policy changes, errors) will be stored in an audit log (e.g. in a database or append-only log store). We will expose this via secure API and the UI. In a multi-tenant setting, each tenant's admin can review their agents' logs. We'll also consider integrations with SIEM tools (Security Information and Event Management) so organizations can incorporate these logs into their centralized monitoring. The delegation token itself may carry an **audit hook** – e.g. a claim with a URL or identifier that a resource service can use to POST back a record of what was donear5iv.org. For example, if an agent uses a delegation token at a service, the service might call back to our logging endpoint with the token ID and action taken.
- **Exception Handling & Revocation:** If an agent attempts something outside its allowed scope (as determined by the policy engine or by a resource rejecting a request), the system will log the incident and could alert the user. In serious cases (e.g. the agent is seemingly compromised or malfunctioning), an automatic trigger could revoke the delegation token to stop further actions. The user or admin would then be alerted to intervene.

Cloud Deployment and Multi-Tenancy

From the outset, we design the library as a **cloud-friendly service**. This includes:

- **Microservices (where appropriate):** The architecture may be broken into services: e.g. *Auth Service* (OIDC + UMA), *Policy Service*, *Audit Service*, *Frontend Webapp*, etc. In an MVP, we might implement it as one application for simplicity, but with clear module boundaries that could be split later. Using containers (Docker/Kubernetes) will allow scaling components independently. For instance, token verification requests might scale differently than the UI usage.
- **Tenant Isolation:** If offered as a SaaS, the system should isolate data by tenant. We can use a tenant identifier in all data models and tokens. The multi-tenant design might leverage either separate instances per tenant (if using something like Keycloak, we can use its "Realm" feature for each tenant), or a single instance with internal tenant scoping. We'll also ensure that administrative UIs and APIs require a tenant admin role to access that tenant's configuration.
- **Scalability and Caching:** Verification of tokens and policy checks should be efficient. We will use caching for token public keys and perhaps use JWTs so that verification is stateless for resource servers. The policy engine should be able to evaluate rules quickly; using OPA, for example, we could keep a policy decision cache. The system will be designed to handle potentially thousands of agent actions per second (if many agents operate concurrently) – so horizontal scaling and load balancing will be in scope for later phases.
- **Security and Compliance:** Operating in the cloud means handling sensitive identity and authorization data. We will adhere to best practices: encrypt tokens at rest, use TLS for all communications, implement proper OAuth token binding (so tokens can't be reused by others), and follow compliance standards (GDPR for personal data, etc.) if this becomes a product. We'll also sandbox any LLM used for policy translation to ensure it doesn't leak sensitive info from the prompt.

Frontend User Interface

The library will be primarily back-end code, but to be MVP-complete, we plan a simple **frontend UI** (likely a single-page application using React or Vue). Key UI elements:

- **Agent Management:** A screen to register AI agents. The user might input an agent name and description, perhaps an API key if the agent is an external service (for example, an OpenAI API key if the agent uses OpenAI under the hood – so the system knows how the agent operates). The UI would display the Agent ID token details (unique ID, metadata).

- **Delegation Editor:** When creating or editing a delegation, the user would see a form to specify what the agent is allowed to do. This could be a multi-step wizard:
    1. **Select Agent and Scope** – choose the agent and choose from predefined permissions (or enter custom instructions).
    2. **Natural Language Input** – an optional text box like "Describe what the agent can do", which our system will parse.
    3. **Policy Preview** – show the structured policy that will be applied (the user can toggle an advanced view to edit JSON/YAML if they are technical, or just trust the defaults).
    4. **Issue Token** – when confirmed, the UI calls the backend to issue the delegation token. The resulting token (or a link/QR code for it) can be shown to the user to give to the agent. For a seamless experience, if the agent is integrated with our system (say running on the same platform), the token could be delivered to it directly.
- **Audit Dashboard:** A table or timeline of recent agent actions. It would list entries like "[Time] Agent X did Y on Service Z – ✅ Allowed by policy" or "❌ Blocked". The user or admin can filter by agent or date, and revoke tokens or change policies in response.
- **Alerts/Notifications:** The UI might highlight if any agent attempted unauthorized actions or if any tokens are expiring soon. We can also integrate email notifications for critical events (for example, "Agent attempted unauthorized action and was blocked").

The UI will use the same REST APIs the public would, ensuring that everything is also scriptable for developers who may not use the UI.

Technical Roadmap (Milestones)

We will implement the above in stages, delivering incremental value:

Phase 1: Core Delegation Framework (Month 1-2)
Goal: Establish the fundamental token system and a basic identity service.

- **Literature & Standard Review:** (Week 1) Finalize the design by reviewing OAuth2/OIDC extension points and UMA specifics. Ensure we align with any emerging standards for "AI agent identity."
- **Identity Provider Setup:** (Week 1-2) Deploy an open-source OIDC server (e.g. Keycloak) and configure custom fields for Agent ID. Implement user login and a simple agent

registration (this might be as straightforward as creating a client credential in Keycloak for the agent).

- **Custom Token Development:** (Week 3-4) Write code (possibly a Keycloak plugin or a standalone microservice using JWT libraries) to issue **Agent-ID tokens** and **Delegation tokens**. Initially, this can be done via a REST endpoint: the user provides their user token (or is authenticated in session), and an agent ID, plus a simple scope list; the service responds with a signed JWT delegation token containing those details. Make sure token verification logic is in place (i.e., a function to validate a delegation token's signature and claims).
- **Basic Scope Enforcement:** (Week 4) Define a simple schema for scopes (e.g. just a list of allowed service:action strings). At this stage, we can hard-code some examples (like "ALLOW:calendar.read"). The token will carry this in a claim.
- **Demo Use-Case:** (Week 4) Create a minimal demo: perhaps a dummy resource server that checks for a valid delegation token. For example, a mock "Web API" that an agent might call – it will accept a delegation token and verify that it allows the requested action. This demonstrates end-to-end: user gets token for agent, agent calls service with token, service verifies and permits.

Phase 2: Policy Engine & NLP Integration (Month 3-4)
Goal: Add the natural language permission translation and a robust policy format.

- **Integrate Policy Engine:** (Week 5-6) Introduce OPA or another policy evaluator. Define our policy model (could be an adaptation of XACML or a custom JSON as decided). Implement a service component that given an agent ID and an action (verb, resource), returns ALLOW/DENY based on the stored policy. Connect this with the token issuance: when issuing a delegation token, also store the full policy server-side (or encode it in the token if small).
- **Natural Language Parsing:** (Week 6-8) Develop the NL->Policy translator. To start, we might implement a rule-based parser for a limited grammar (e.g., detect keywords like "can/cannot", named objects like "Salesforce accounts", etc.). If feasible, we experiment with an LLM by prompting it with examples to generate structured policies (with the user confirming results). The output is then fed into the policy engine. Ensure that any free-form parsing is always confirmed by the user (to mitigate errors).
- **User Consent & Preview:** (Week 8) Implement a mechanism for the user to preview the derived policy (maybe return it in the API response for token issuance before finalizing). This might be done via the UI in Phase 4, but we can simulate it via API for now.

- **Test with Complex Scenarios:** (Week 9) Write unit tests for the policy translator with various phrasing. Also test the policy enforcement with edge cases (e.g., overlapping rules, denial overrides). We might also incorporate example policies from standards (like an XACML sample) to ensure our format is expressive enough.

Phase 3: Secure Integration and Agent SDK (Month 4-5)
Goal: Enable real-world interactions and strengthen security.

- **Third-Party API Connectors:** (Week 10-12) Implement one or two real integrations. For instance, **Google Calendar integration**: set up an OAuth app, and implement a flow where our service, upon receiving a delegation token with "calendar" scope, exchanges it for a Google OAuth token (this might involve the user pre-authorizing our app with Google). Document this process. Similarly, attempt a **GitHub integration** (an agent could create an issue on GitHub on user's behalf) to showcase a different domain. These connectors will likely involve storing credentials securely (encrypted refresh tokens) and careful scope mapping.
- **AI Agent SDK:** (Week 10-13) Create a lightweight SDK for AI agents to use the delegation framework. For example, a Python library that an agent (perhaps built with LangChain or similar) can use to request a delegation token and present it when calling APIs. This SDK might provide helper functions like agent.login_via_delegation(server_url) which opens a browser for the user to approve the agent, etc., or simpler if tokens are provided out-of-band. The idea is to make it straightforward for developers building AI agents to integrate our library – they shouldn't have to craft JWTs manually.
- **Security Review & Hardening:** (Week 14) Conduct a thorough review of the security model. Ensure tokens have proper audience restrictions (so they can't be replayed to unintended services), consider using **Proof-of-Possession** tokens (bound to agent's key) so that if a token leaks, it's unusable without the agent's private key. Mitigate **prompt-injection** risks by warning that the agent LLM should never reveal its tokens (perhaps recommend using retrieval augmentation so the LLM doesn't directly see the raw token, instead the agent code handles it). If possible, implement token hashing or partial redaction when the LLM is handling it. At this phase, also implement **logging** of all token issuances and accesses in a secure log.

Phase 4: Full MVP Implementation (Month 6)
Goal: Deliver the end-to-end product experience with UI and multi-tenant deployment.

- **Frontend UI:** (Week 15-16) Develop the React/Vue frontend as described. Use the backend APIs to list agents, create delegation tokens, etc. Implement authentication for the UI (likely the user logs into our IdP to use the UI – since it's OIDC, we can protect the

UI routes for authenticated users). Ensure the UI is mobile-friendly if possible, as users might want to approve requests on the go.

- **Multi-Tenant Support:** (Week 17) Expand the system to handle multiple organizations. This could involve adding a tenant ID field to relevant database tables, and an admin management API to create new tenants. We'll test by simulating two tenants with separate users/agents and ensure no crossover. If using Keycloak, each realm could serve as a tenant; we'd then spin up the delegation service per realm or adapt it to realm-specific endpoints. Document how a company could deploy this for their internal use (possibly offering a Helm chart or docker-compose file for quick setup).
- **Documentation & Examples:** (Week 18) Finalize documentation: how to run the service, API docs (Swagger/OpenAPI specs), and example code for using the SDK. Provide a tutorial scenario (e.g. "Delegate an agent to post on Slack for you") with step-by-step instructions. Ensure all citations or references in our code (if any standards used) are credited appropriately.
- **Alpha/Beta Release:** (Week 19-20) Publish the code repository (likely on GitHub) with an open-source license (to be determined, possibly Apache 2.0 for broad use). Do an internal or closed-group beta test with a few developers, incorporate feedback on usability, fix bugs. Prepare a roadmap for future improvements (see below).

Example Use Cases and Demo Scenarios

To illustrate the functionality of the library, consider a few realistic scenarios:

- **1. AI Sales Assistant for CRM:** A sales manager delegates an AI agent to help manage their Salesforce data. Using our framework, the manager issues a delegation token that permits the agent to **read contacts and create draft opportunities** in Salesforce, but *not* to delete anything or send communications. The agent (which could be built on GPT-4 with access to Salesforce API) then uses the provided credentials to perform tasks like compiling a daily report of new leads and adding notes to contact records. Whenever the agent tries to do something, Salesforce's API (or our proxy in between) checks the delegation token's scope. If the agent attempts an unauthorized action (e.g., deleting a contact), the request is denied and logged. The manager can later audit all the changes the agent made, with each change tied to the agent's identity and the delegation token ID for traceability.
- **2. Project Management Agent:** A team uses an AI agent to triage and organize tasks in Jira and Trello. The human team lead gives the agent permission to **create and comment on Jira issues in Project XYZ, and move Trello cards between columns**, but not to close tasks or change settings. The agent obtains an agent ID token and a delegation token

encoding these permissions. It then runs periodically: reading new issue descriptions (via Jira API), summarizing them, and creating corresponding Trello cards. The integrations use our library's connectors – for Jira, the agent's delegation token is exchanged for a Jira API token limited to Project XYZ. For Trello, our library uses an API key/token with only board-specific access. All actions carry the delegation token reference so that Jira and Trello logs can indicate the AI agent (acting for user Jane Doe) did X action. If the agent is unsure about a task (perhaps the instructions are unclear), it can flag it for human review. The team lead can adjust the agent's policy on the fly (e.g., give it permission to assign issues to certain team members by updating the scope and reissuing a token).

- **3. Personal Executive Assistant:** An individual user employs a GPT-based agent to handle personal tasks: managing calendar, sending emails, and paying bills online. This is high risk, so the user configures tight controls. Through the UI, they allow the agent to **read** their email and **draft replies**, but any outgoing email must be approved. They also allow it to **schedule appointments** on their Google Calendar, and to **make payments up to $100** on a specific bills website. The agent gets a delegation token that, for email, perhaps grants access to the Gmail API for reading and creating drafts (but not sending), and for payments, a token that the bill-pay API will accept for limited transactions. Whenever the agent tries to actually send an email or make a payment, our library intercepts the action and sends a notification to the user: "Agent wants to send email to X. Allow?" The user can approve via the mobile UI. This human-in-loop step is logged. Over time, if the user trusts the agent more, they could widen the policy (or the system could learn from the user's approvals to grant more autonomy, always under user control).

These scenarios demonstrate how the library can be used in both enterprise and personal contexts to enable powerful AI assistance with maintained security. They also highlight the importance of each component: identity assurance (knowing it's the agent acting), policy enforcement (the agent stays in bounds), and auditability (we can always verify what happened and by whom).

Challenges and Risk Mitigation

While building this framework, we must address several challenges:

- **Security of AI Agents:** Unlike traditional apps, LLM-based agents might not be entirely deterministic or secure. There is a risk of **prompt injections** or the agent being manipulated to reveal or misuse its credentials. Our mitigation: minimize the exposure of raw tokens to the LLM. For example, the agent's code (outside the model) should handle inserting tokens into API calls, rather than the model having the token in its text prompt. We also enforce that tokens are scoped minimally, so even if leaked, they can only do a narrow set of actions. Prompt injection attempts that make an agent step outside its scope will fail because the external APIs will deny unauthorized actions – effectively containing the damage. Additionally, using techniques like monitoring the agent's outputs for any presence of the token (via regex) and immediately revoking it could provide an extra safety net.

- **Adoption by Third-Party Services:** For this system to work widely, external services (like Salesforce, Google, etc.) need to trust and use the delegation tokens or the derived credentials. Initially, we handle this by acting as a proxy or using their OAuth2 on-behalf flows. This means our library itself will act as the integrator. In the long run, advocacy for an open standard is key – e.g., if our framework (or the underlying research) becomes standardized, services could natively support verifying delegation tokens. Until then, we mitigate the adoption issue by making integration as invisible as possible (packaging the necessary logic in our adapters). Using widely accepted protocols (OIDC, UMA, JWT, etc.) also increases the chance of acceptance – we're not creating an entirely new auth scheme, but building on known ones.
- **Privacy and Data Handling:** AI agents might deal with sensitive user data. Our library will be in a privileged position, managing tokens that could access emails, files, etc. It's critical to ensure data is protected. All tokens and policies will be stored encrypted at rest. We won't store user content (like email text) on our servers except transiently if needed for policy parsing (and that can be done client-side or in-memory). We also allow self-hosting of the solution for those who require full control (the open-source nature means a company could run the delegation service on their own infrastructure so that no third party – not even us – ever sees their data). We'll provide guidelines in documentation for secure deployment (e.g. rotate encryption keys, monitor logs for anomalies).
- **Complexity of Natural Language Understanding:** Accurately interpreting a user's intent for agent permissions is non-trivial. There's a risk of misinterpretation leading to over-broad or over-narrow permissions. To mitigate this, our policy editor will always show the **explicit resulting permissions** for confirmation. We might also incorporate a library of **predefined roles** (like templates: "Read-Only Assistant", "Can Send Email" etc.) to choose from, which are less error-prone than free text. Over time, we expect the NLP model to improve with feedback (e.g., if users repeatedly correct a certain policy suggestion, we adjust the parsing rules or training data). During the MVP, we will keep the scope of NL instructions limited and well-tested to avoid misunderstandings.

- **Performance and Scalability:** If an agent is performing actions in a loop, adding our checks could introduce latency (each action needing a token check or exchange). We mitigate this by using JWT self-contained tokens where possible (so a service can validate quickly without network calls). Also, caching access tokens for third-party services for short durations can reduce repetitive exchanges (with careful observance of scope). We will benchmark the overhead; our aim is that using the delegation framework adds minimal latency (~tens of milliseconds per call for verification). If we find bottlenecks, we might offload some checks to a faster in-memory service or use asynchronous processing for audit logging to not slow down the agent.
- **Legal and Compliance Questions:** Delegating authority to an AI has legal implications (for instance, who is responsible if the agent causes harm?). While the library itself cannot solve legal liability, it provides the **accountability trail** that is often needed – one can always identify which human authorized the agentfile-tzfekfrsucjnj2cztgs6es. We will include features like requiring the user to periodically re-confirm the delegation (to ensure they remain aware of what the agent is doing). Additionally, by limiting scope, we help prevent an agent from doing things that a user would not legally allow. As this project could evolve into a startup, we will also consult with legal experts on features like Terms of Service management (maybe embedding a terms acceptance in the delegation process).

Future Outlook

Upon completing the MVP, the next steps would involve expanding and refining the system based on real user feedback. A likely enhancement is support for the **W3C Verifiable Credentials (VC)** model as an alternative to JWT tokensar5iv.org. VCs could make the delegation credentials usable in a decentralized context (e.g., an agent could operate across domain boundaries, with each party verifying a signed credential instead of calling back to a central IdP). This aligns with the paper's suggestion that VCs can be used to convey the delegation in a portable, decentralized mannerfile-tzfekfrsucjnj2cztgs6es. We anticipate adding VC support as an interoperability option (possibly converting a delegation JWT into a VC format).

Another area is **cross-domain agent federation** – enabling agents from different organizations to collaborate by verifying each other's delegation credentials (the paper's Figure 4 touches on this). Our roadmap includes exploring federated trust, where multiple IdPs trust a common standard for agent credentials.

Finally, as adoption grows, we would seek to contribute this framework to standardization bodies (IETF OAuth extensions or W3C), so that the concept of "Authorized AI Agents" becomes an open standard across the industry. This would reduce the burden on each service to integrate separately and truly unlock the value of safe, accountable AI delegationar5iv.orgfile-tzfekfrsucjnj2cztgs6es.

In conclusion, this project plan lays out a comprehensive path to build a first-of-its-kind library for authenticated AI delegation. By combining proven identity protocols with novel agent-specific extensions, we aim to enable **trustworthy AI agents** that can take actions for users in the real world – but only **with the user's identity attached and their privileges constrained**. This approach will protect online systems from rogue AIs while empowering beneficial automation, striking a balance between innovation and securityar5iv.orgar5iv.org. The open-source library will allow the community to experiment, audit, and improve the framework, ultimately guiding it from an MVP into a robust platform (and potentially, the foundation of a new startup or standard) for the age of AI-driven services.

Sources: The design is informed by the framework proposed by South *et al.* (2025)ar5iv.orgar5iv.org and utilizes existing standards like OAuth 2.0, OIDC, and UMAar5iv.org to ensure compatibility. The challenges and solutions discussed align with the need for auditable, scoped delegation to AI agents as highlighted in that workfile-tzfekfrsucjnj2cztgs6esfile-tzfekfrsucjnj2cztgs6es. All technical components will be built with community-vetted open-source technologies to accelerate development and adoption.

Citations

2501.09674v1.pdf

file://file-TzFEkFRsUCjnJ2CZTgS6Es

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

2501.09674v1.pdf

file://file-TzFEkFRsUCjnJ2CZTgS6Es

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

2501.09674v1.pdf

file://file-TzFEkFRsUCjnJ2CZTgS6Es

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

2501.09674v1.pdf

file://file-TzFEkFRsUCjnJ2CZTgS6Es

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

2501.09674v1.pdf

file://file-TzFEkFRsUCjnJ2CZTgS6Es

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

[2501.09674] Authenticated Delegation and Authorized AI Agents

https://ar5iv.org/pdf/2501.09674

All Sources

2501.09674v1.pdf

ar5iv