# Standard OAuth 2.0 Delegation Flow Outline (On-Behalf-Of)

This outline details a vendor-agnostic implementation design for authenticating a user (Client 1), delegating that user's authority to an intermediate service (Client 2), and ensuring the downstream component (Resource Server) can validate the token based on open standards (OIDC/RFC 8693).

## Key Components and Terminology (Standards-Based)



| **Component Name** | **Client ID** | **OAuth Role** | **Description** | **Okta Implementation Note** |
| --- | --- | --- | --- | --- |
| **Client 1** | contextflow | Delegator (Initial App) | The application where the user logs in and obtains the Subject Token. | Registered as an OIDC Application (Web/SPA/Native). |
| **Client 2** | mcp-oauth | **Actor / Requesting Client** | The intermediate component (e.g., your sub-system) that performs the token exchange. | Registered as an **API Service** application, granting it an identity for OBO flow. |
| **Resource Server** | N/A (Endpoint) | Resource Server | The MCP Sub-component that protects the target API endpoint. | The service that validates the token signature and claims. |
| **Authorization Server** | N/A | Issuer (iss) | The central authority minting tokens. | A **Custom Authorization Server** must be used (not the Org Authorization Server). |

## Phase 1: User Authentication and Subject Token Acquisition

This phase uses the standard OAuth 2.0 Authorization Code Flow (with PKCE for maximum security).

### 1. Client 1 (contextflow) Setup

* **Grant Type:** Authorization Code Flow (required).
* **Permissions:** Request scopes like openid profile email.
* **Audience Scoping (Crucial for Delegation):** Client 1's token configuration **MUST** be set up to include the Client 2's ID (mcp-oauth) in the aud (Audience) claim of the Subject Token.

| **Okta Implementation Steps for Audience Scoping** |
| --- |
| **1. Define Scope on Auth Server:** Create a **Scope** (e.g., api:mcp:exchange) on the Custom Authorization Server. |
| **2. Define Claim:** Create a **Claim** on the Auth Server that is included in the **Access Token** and maps a group/user attribute, or simply sets the aud claim based on a policy. |
| **3. Trust Configuration (Explicit):** On the Client 1 application settings, the Auth Server's **Access Policy** must have a **Rule** that grants Client 1 access to the mcp-oauth resource, ensuring the resulting token includes the required audience to permit the exchange. |

### 2. User Flow

1. User initiates SSO login via Client 1 (contextflow).
2. IDP authenticates the user and issues the **Subject Token** (JWT).
3. **Subject Token Claims (Expected):**
   * aud: ["contextflow", "mcp-oauth", ...]
   * azp: "contextflow"
   * sub: <User ID>

## Phase 2: Token Exchange (Delegation / On-Behalf-Of)

This phase is executed by the backend of Client 2.

### 1. Client 2 (mcp-oauth) Setup

* **Client Type:** **Confidential** (required for secure server-to-server exchange).
* **Grant Type:** **Token Exchange** (urn:ietf:params:oauth:grant-type:token-exchange) enabled.
* **Delegation Permission:** Client 2 must be configured to be allowed to receive a delegated token.

| **Okta Implementation Steps for Delegation Permission** |
| --- |
| **1. Enable Token Exchange Grant:** On the Client 2 application (mcp-oauth), navigate to the **General Settings** and explicitly enable the **Token Exchange** grant type. |
| **2. Policy Access:** Ensure the **Access Policy** on the Custom Authorization Server grants Client 2 (mcp-oauth) permission to use the Token Exchange grant. |

### 2. Token Exchange Request

Client 2 sends a POST request to the IDP's /token endpoint:

| **Parameter** | **Value (Standards-Based)** |
| --- | --- |
| grant\_type | urn:ietf:params:oauth:grant-type:token-exchange |
| client\_id | mcp-oauth |
| client\_secret | <Secret of mcp-oauth> |
| subject\_token | <Subject Token JWT> |
| subject\_token\_type | urn:ietf:params:oauth:token-type:access\_token |
| **audience** | **mcp-oauth** |
| scope | <Necessary Scopes> |

### 3. Exchanged Token Claims (Expected)

The IDP issues the **Exchange Token** (JWT), which proves delegation:

* **aud**: ["mcp-oauth", ...] (Audience for the recipient service.)
* **azp**: "mcp-oauth" (The **Actor** who requested the token.)
* **act**: The optional but highly recommended **Actor Claim** (RFC 8693) will contain details about the original subject (the user).
* sub: <User ID> (The identity remains the user's).

## Phase 3: Resource Server Validation (The Core Security Fix)

The MCP Sub-component (Resource Server) performs validation that is portable across any compliant IDP.

### 1. Standard JWT Validation

The Resource Server performs baseline checks:

* Verify signature using IDP's public key (retrieved from the Auth Server's JWKS endpoint).
* Verify expiration (exp).
* Verify issuer (iss) matches the IDP's Custom Authorization Server URL.

### 2. Delegation and Source Validation (The Security Solution)

The Resource Server checks two claims to ensure the token is the correct, exchanged token, and not the original:

| **Claim to Check** | **Expected Value** | **Security Purpose (Vendor-Agnostic)** |
| --- | --- | --- |
| **aud (Audience)** | Must contain "mcp-oauth". | Ensures the token was intended for this service. |
| **azp (Authorized Party)** | **MUST** exactly equal "mcp-oauth". | **This is the critical check.** It proves the token was minted specifically for the intermediate service (Client 2) to act as the current actor, rejecting the original user's token (azp: contextflow). |

**Conclusion:** By checking the **azp** claim, the MCP Sub-component implements a robust, standards-based security policy that solves the vulnerability of accepting the high-privilege Subject Token. The implementation steps, while different in Okta (relying on **Custom Authorization Servers** and **Access Policies**), map directly to the same required OAuth concepts defined in the initial Keycloak flow.