**Secure Workload Identity Federation (WIF) Token Distribution with Mutual TLS**

**1. Objective**

This white paper documents the secure, automated generation and distribution of short-lived Workload Identity Federation (WIF) tokens using JWT, Azure AD, Google's STS, and mutual TLS (mTLS) to ensure secure delivery.

**2. Background**

Workload Identity Federation (WIF) allows applications outside Google Cloud to securely impersonate Google Cloud service accounts using third-party identities (e.g., Azure AD).

Instead of managing long-lived service account keys, this solution uses:

* JWTs signed with private keys
* Token exchanges via STS
* Short-lived OAuth tokens

This Java-based utility automates WIF token generation and exposes it via a Flask-based HTTPS server using mTLS.

**3. Architecture Overview**

**🔧 Key Components:**

* **Java WIF Token Generator**:
  + Reads config from app.properties, keystore.json, and provider.json
  + Signs JWT using provided private key
  + Sends request to Azure AD to get OIDC token
  + Exchanges it via Google's STS for a GCP access token
  + Saves token in wif\_token.txt
* **Flask HTTPS Server with mTLS**:
  + Serves the access token securely via /get\_wif\_token
  + Requires mutual TLS authentication
* **Client Python Script**:
  + Performs mutual TLS handshake using client.crt and client.key
  + Downloads token securely
  + Saves it locally for use in GCP-bound applications

**4. DevOps Onboarding Workflow**

|  |  |  |
| --- | --- | --- |
| **Step** | **Description** | **Output** |
| 1 | Generate certificate via internal PKI | CRT, PKCS12 Keystore |
| 2 | Request private key via cdkeyprotect team | CSR file |
| 3 | Sign and share public cert back to cdkeyprotect | Signed CRT |
| 4 | Azure AD onboarding | Tenant ID, Client ID |
| 5 | Extract sub value from token | sub claim in JWT |
| 6 | GCP service account creation | GCP SA Email |
| 7 | WIF Pool mapping | WIF Pool + sub claim + SA ID |

**5. Security Enhancement: Mutual TLS**

**🛡 Why mTLS?**

* Prevents token endpoint from being accessed by unauthorized clients
* Ensures only clients with valid signed certs can access /get\_wif\_token

**🔐 Server Setup**

* Uses server.crt, server.key, and a ca-chain.crt (with root + intermediate)
* Flask app configures SSLContext with CERT\_REQUIRED

**🔐 Client Setup**

* Uses client.crt, client.key, and same ca-chain.crt
* Uses Python requests.get(..., cert=(crt, key), verify=ca)

**🔐 Certificate Chain Validation**

* Server and client certificates must be signed by **matching CA hierarchy**
* Mismatched chains (e.g., CA 13 vs CA 13.1) will fail

**6. Technical Highlights**

* No use of long-lived service account keys
* Automatic refresh every 55 minutes
* JWT signing and validation with secure private key
* Uses Google's STS Token Exchange flow
* mTLS ensures confidentiality and client authenticity

**7. Example Timeline**

📈 DevOps: Create Certs

⬇

📈 Java App: Generate WIF JWT

⬇

📈 Exchange via STS

⬇

📈 Flask App: HTTPS + mTLS

⬇

📈 Client: mTLS Token Fetch

**8. Future Enhancements**

* Use short-lived client certs from Vault or internal PKI
* Add token signing validation audit logs
* Optionally support OAuth2 or bearer tokens for fallback environments

**9. References**

* Google STS for WIF
* [Azure AD OIDC token endpoint](https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-protocols-oidc)
* [Mutual TLS in Python](https://docs.python.org/3/library/ssl.html)