WOKRING OF FLOW DIAGRAM:

### **Step 1: User Login (Authentication)**

* **Component Involved**: Native App Tool → IAM (OIDC) → Identity Provider (IdP)
* **Protocol**: OIDC (OpenID Connect over HTTPS)

#### **Flow:**

1. **User opens app**, selects diagnostic service.
2. App uses OIDC to redirect user to login.
3. Login credentials are submitted to **Identity Provider (LDAP/SSO)**.
4. If successful, **IAM (Cognito)** returns:
   1. id\_token (user identity)
   2. access\_token (authorization token)
   3. refresh\_token (for future use)

#### **Commands Used:**

http

CopyEdit

POST /oauth2/token  
  
  
grant\_type=password  
[&username=user@example.com](mailto:&username=user@example.com)&password=abc123  
&client\_id=CLIENT\_ID  
&scope=openid

### **🧩 Step 2: Token Validation (Authorization)**

* **Component Involved**: Native App Tool → Cloud MAM
* **Protocol**: HTTPS + Custom REST APIs

#### **Flow:**

1. App sends access\_token to **MAM**.
2. MAM verifies token by contacting **IAM (OIDC introspection)**.
3. MAM parses the token to extract:
   1. user\_id
   2. roles
   3. validity

#### **Commands Used:**

http

CopyEdit

POST /introspect  
Authorization: Basic base64(client\_id:client\_secret)  
Content-Type: application/x-www-form-urlencoded  
  
token=eyJhbGciOiJSUzI1NiIs...

### **🧩 Step 3: Certificate Validation (OCSP)**

* **Component Involved**: Cloud MAM → OCSP Responder
* **Protocol**: OCSP (Online Certificate Status Protocol)

#### **Flow:**

1. App presents its client certificate (X.509).
2. MAM builds an **OCSP request** for the presented cert.
3. Sends to OCSP responder.
4. Responder replies with cert status: GOOD, REVOKED, UNKNOWN.

#### **OCSP Commands:**

* **OCSP Request Format** (Encoded DER):

bash

CopyEdit

openssl ocsp \  
 -issuer issuer\_cert.pem \  
 -cert user\_cert.pem \  
 -url <http://ocsp.example.com> \  
 -CAfile ca\_cert.pem

* **MAM Calls**:

http

CopyEdit

POST /ocsp  
Content-Type: application/ocsp-request  
  
(binary DER-encoded OCSPRequest)

* **OCSP Response**:

http

CopyEdit

HTTP/1.1 200 OK  
Content-Type: application/ocsp-response  
  
(binary DER-encoded OCSPResponse with status GOOD)

**🧩** **Step 4: License Verification**

* **Component Involved**: Cloud MAM → OMS/CTAS
* **Purpose**: Ensure the user has a valid license to use diagnostic services.

#### **Flow:**

1. MAM sends user\_id and service\_type to **OMS/CTAS**.
2. License Manager responds with:
   1. License valid?
   2. Usage limits
   3. Service authorization

#### **Commands:**

http

CopyEdit

POST /license/validate  
Content-Type: application/json  
  
{  
 "user\_id": "abc123",  
 "service": "remote\_diag",  
 "vehicle\_id": "VIN123456"  
}

json

CopyEdit

Response:  
{  
 "license\_valid": true,  
 "usage\_quota": "unlimited"  
}

### **🧩 Step 5: Logging/Database:**

* Tracks **who did what**, **when**, **from where**, and **on which ECU/tool**.
* Useful for **security audits** and **traceability**.
* Whether the tool had valid permission (validated via OMS/CTAS).
* Records of certificate status and validation timestamps.
* In flow diagram:
  + It sends request to check user certificate and MAM.
  + If it approves it will accept the service and if it rejects it ask to reissue the certifcate

### **🧩 Step 6: Certificate verification**

* Cloud MAM checks if a certificate for a specific identity (e.g., ECU ID, App Tool) **already exists** in the CA’s database.
* Cloud → CA:
  + Request: “Does a certificate exist for this identity (Serial Number or Common Name)?”
* CA → Cloud:
  + Response: **Yes** or **No** (entry found or not found).
* Commands:

GET /certs?serial=123456

Response: {"exists": true}

### **🧩 Step 7: Logging and Database**

* Log the data to database with User,UserID, Certificate, Permissions, User Public key and Private key etc
* It helps to keep track of user in the enviroment.

### **🧩 Step 8: Requesting user productID**

1. **Product ID** is a **unique identifier** embedded in the **ECU firmware** or memory.  
    It represents the ECU’s:
2. Type or model,
3. Configuration,
4. Diagnostic capabilities,
5. Licensing status,
6. Security credentials (sometimes bound to certificates)
7. Flow:
   * Tool->ECU: it ask for product id it require.
   * ECU->tool: Responds with Product ID to identify itself.
   * Uses Product ID to determine supported diagnostic services, link to certificate, and request appropriate license.
   * Maps Product ID to available licenses, certificates, or ECU policies.
   * Product ID might be linked to certificate metadata, to track valid issuers or users.

* MAM\_initDiagAccess: initializes MAM cloud instance of user so MAM is prepared to connect to module with that tool user is logged in on Checks for valid issued cert in DB, goes and gets one if needed, obtains license NOTE-This logic does not currently have "Device binding", but may be the place to add if we want that funcitonality.

### **🧩 Step 9: Requesting user tool certificate:**

The main goal of this message is to **retrieve the digital certificate** that uniquely identifies and authenticates the diagnostic tool. This certificate is used for:

* 🧾 **Authorization** of the tool by Cloud MAM.
* ✅ **Verification** by the ECU before allowing access.
* 🔐 **Secure communication** via mutual TLS (mTLS).
* 🧩 **Binding** the tool identity to licenses and security policies.
* Validates that the tool was issued by the correct OEM Certificate Authority.
* the certificate can be checked via OCSP for revocation status.

Response must contain:

* + **Response: Tool Certificate (X.509)**  
     Cloud MAM (or local store) responds with the full certificate, which includes:
  + Public key,
  + Validity period,
  + Serial number,
  + Subject name (Tool ID, Product ID),
  + Issuer (CA),
  + OCSP URL (for validity check).
* MAM\_initECM: gets updated certs from MAM (if needed) & tells MAM what ECM will be connecting

### **🧩 Step 10: OCSP**

* **OCSP** = **Online Certificate Status Protocol**
* It is a protocol used to check whether a **digital certificate** (used for secure communication or authentication) is **still valid**, **revoked**, or **expired** — *in real time*.
* Instead of downloading a full Certificate Revocation List (CRL), the client makes a lightweight HTTP request to an OCSP responder and asks:
* “Is certificate X still valid?”
* UPDATEOCSPRESPONSE: The App Tool is requesting an **OCSP response update** — it wants **fresh certificate status**.
* NONCE: A **nonce** (number used once) is a **random value added to the OCSP request**.
  + Prevent **replay attacks**.
  + Ensure the response is **fresh** and **matches the request**.
* It send ocsprequest to CA to verify certificate
* It get oscpresponse from CA it check the certificate with the ecu.
* MAM\_getEph Key: gets an ephpubkey from MAM that is unique and associated with a specific connection instance
* Then it store the session into log to keep track of its working in diagnostics.

### **🧩 Step 11: MAM verification**

* MAM\_getbidirVerifyCert:
* The **Tool's certificate** (X.509),
* A **random challenge** from MAM (nonce or timestamp),
* Instructions on how to verify ECM's certificate later.
* This is the **first step** of a **bi-directional certificate verification** flow. It ensures both the **Tool and ECM** can **prove their identity and ownership of private keys**.
* MAM\_putbidirVerifyCert:
* Handles the **response** from the ECM and completes the **cryptographic checks**:
* Validates ECM’s certificate.
* Validates ECM’s **proof of ownership** (signature on challenge).
* Then generates the **Tool’s proof of ownership** based on the ECM's challenge (mutual verification).
* MAM\_getSessionKey:
* Once mutual authentication is complete, this API provides the **session key** for secure diagnostics.
* Session key is symmetric (e.g., AES) and used for:
* Encrypting diagnostic communication
* Ensuring integrity and confidentiality between Tool and ECM

### **🧩 Step 12: Proof of Ownership Request & Response**

* **Proof of Ownership** (PoO) is a **security mechanism** used to verify that a device or tool **possesses the private key** corresponding to its public certificate. This ensures the component is **genuinely the entity** it claims to be.
* This process is **bidirectional** between the **Tool** and the **ECM (ECU)**.

### **ECM Proves Ownership**

#### **Request**

* The **Tool sends a challenge** to the **ECM** during verifyCertificateBidirectional.
  + This challenge could be a random number or nonce from MAM\_getbidirVerifyCert.

#### **Response**

* ECM responds with:
  + Its **X.509 certificate**
  + A **signature** of the challenge using its private key  
     *(Proof of Ownership = Sign(Challenge))*

This shows ECM owns the private key corresponding to its public cert.

### **🔹 2. Tool Proves Ownership**

#### **Request**

* The **ECM sends a challenge** (its own random nonce) to the **Tool** after validating the tool certificate.

#### **Response**

* The **Tool signs the ECM’s challenge** with its private key and sends back:
  + The **signed challenge** → This is Tool’s Proof of Ownership.

This proves the Tool holds the private key corresponding to its certificate.