Chapter 4 — Architecture and Authentication

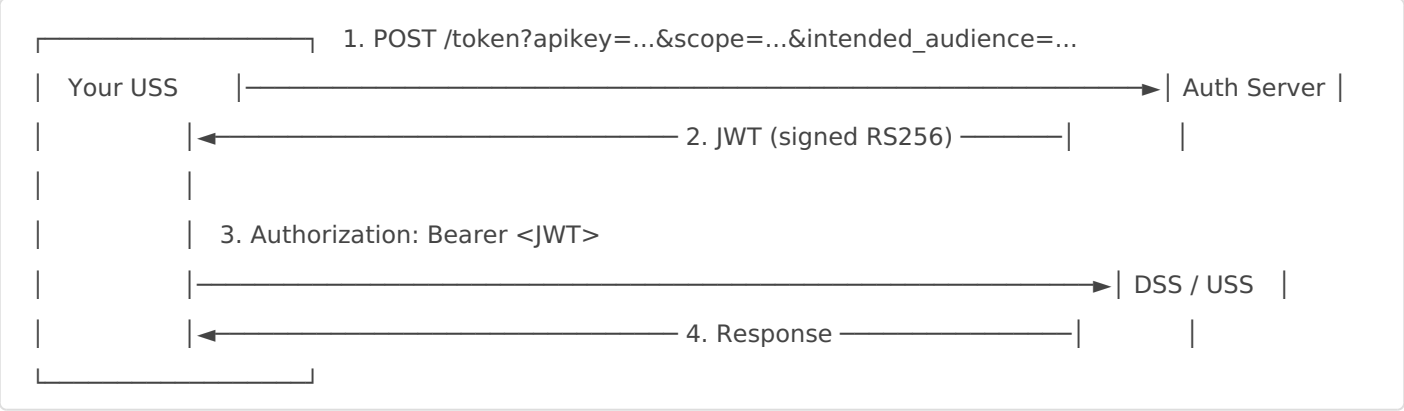
This chapter describes the authentication and authorization model used across the entire BR-UTM ecosystem, including how your USS obtains tokens, uses them to call DECEA's APIs, validates tokens received from other USSs, and authenticates peer-to-peer calls.

4.1 Authentication Architecture Overview

BR-UTM uses **OAuth2 Client Credentials flow** with **JWT (JSON Web Token) access tokens**, signed using **RS256 (RSA SHA-256)**. This is a stateless, distributed authentication model where:

1. Your USS presents its **API Key** to DECEA's **Auth Server** to obtain a signed JWT.
2. The JWT is included as a **Bearer token** in every API request (to the DSS, to DECEA's USS, or to other USSs).
3. The receiving server **validates the JWT locally** using DECEA's public key — it does not need to call the Auth Server again.

This architecture is critical for the distributed, peer-to-peer nature of the system: there is no central session state, and any server can independently validate any token.



4.2 Obtaining a JWT Token

To obtain a JWT, send an HTTP GET or POST request to the Auth Server token endpoint:

Endpoint: `http://api.sandbox.brutm.dcta.mil.br/token`

Query parameters (or request body):

Parameter	Description	Example
`apikey`	Your company's API Key, obtained from the Portal UTM. Can also be sent as an HTTP header.	`abc123...`
`scope`	The OAuth2 scope(s) you need for the operation. Space-separated for multiple scopes.	`utm.strategic_coordination utm.constraint_processing`
`intended_audience`	The FQDN (domain name only, no path) of the server this token will be sent to.	`api.sandbox.brutm.dcta.mil.br` or `uss-b.yourpartner.com`

Example request:

```
GET /token?scope=utm.strategic_coordination&intended_audience=api.sandbox.brutm.dcta.mil.br
Authorization: ApiKey abc123yourapikey
```

Example response:

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

4.3 JWT Token Structure

Every token issued by DECEA's Auth Server is a signed JWT containing the following claims:

Claim	Description	Example
`iss`	The URL of the Auth Server that issued the token.	`http://api.sandbox.brutm.dcta.mil.br/token`

Claim	Description	Example
`exp`	Token expiration timestamp (Unix epoch). Maximum 1 hour from issuance.	`1751327400`
`sub`	Your USS's unique identifier — the `manager` identifier in DSS records.	`"my-company-uss"`
`scope`	Granted scopes, space-separated.	`"utm.strategic_coordination utm.constraint_processing"`
`jti`	Unique token ID for replay protection (RFC 7519).	`"d3e8f921-..."`
`aud`	The FQDN of the intended recipient server.	`"api.sandbox.brutm.dcta.mil.br"`

Token Validity

- Tokens are valid for **up to 1 hour** (inspect the `exp` claim for the exact expiry time).
- **Cache and reuse tokens** until they are near expiry. Requesting a new token on every API call is wasteful and may cause rate limiting.
- A single token can carry **multiple scopes** (space-separated in the `scope` claim), so you can request all the scopes you need for a workflow in one token request.

4.4 Available Scopes

UTM API (`utm.yaml`)

Scope	Purpose
`utm.strategic_coordination`	Create, update, delete, and query OIRs in the DSS. Notify subscriber USSs. Fetch OIR details from other USSs. **Required for all standard flight operations.**
`utm.constraint_processing`	Query constraint references in the DSS and fetch constraint details from DECEA's USS. **Required if your area may have constraints.**
`utm.conformance_monitoring_sa`	Query OIRs and fetch telemetry for off-nominal situations. Used by DECEA's monitoring systems.
`utm.constraint_management`	Create, update, and delete constraints. **DECEA only.**
`utm.availability_arbitration`	Set USS availability state in the DSS. **Not required in Phase 1.**

Scope	Purpose
`utm.aviation_authority`	Access flight authorization details. **DECEA only.**

Remote ID API (`remoteid.yaml`)

Scope	Purpose
`rid.service_provider`	Create, update, and delete ISAs in the Remote ID DSS. **Required when activating/deactivating flights.**
`rid.display_provider`	Query ISAs and telemetry from other USSs. **Required for situational awareness.**

Automated Testing APIs

Scope	Purpose
`interuss.flight_planning.direct_automated_test`	Used by DECEA's test framework when calling your `flights.yaml` endpoints during homologation.
`interuss.flight_planning.plan`	Used by DECEA's test framework to simulate user flight plan actions.
`rid.inject_test_data`	Used by DECEA's test framework when calling your `injection.yaml` endpoints during homologation.
`interuss.versioning.read_system_versions`	Used by DECEA's test framework when calling your `versioning.yaml` endpoint.

Day-to-Day Scope Requirements

For normal production operations, your USS will primarily need:

```
utm.strategic_coordination
utm.constraint_processing
rid.service_provider
rid.display_provider
```

4.5 The `intended_audience` Parameter — Peer-to-Peer Calls

The `intended_audience` parameter (which becomes the `aud` claim in the JWT) is **critical for peer-to-peer security**. It binds a token to a specific recipient, preventing token replay attacks.

Rules:

- When calling **DECEA's DSS** or **Auth Server**: use the domain of the DSS (e.g., `api.sandbox.brutm.dcta.mil.br`).
- When calling **another USS** (e.g., to fetch OIR details): use **only the domain** of that USS's `uss_base_url` — no path, no port (unless non-standard).

Example:

If USS B's `uss_base_url` in the DSS is `https://uss-b.partnercompany.com/api/utm`, then when USS A wants to call USS B:

```
intended_audience = "uss-b.partnercompany.com"
```

USS A requests a fresh token with this audience and includes it in the `Authorization` header when calling `GET https://uss-b.partnercompany.com/api/utm/uss/v1/operational_intents/{id}`.

USS B, upon receiving this request, validates that the token's `aud` claim matches its own domain (`uss-b.partnercompany.com`). If it doesn't match, it rejects the request with HTTP 401.

4.6 Using the Token

Include the token in every outgoing HTTP request:

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

This applies to calls to:

- DECEA's DSS (`/dss/v1/...`)
- DECEA's Remote ID DSS (`/dss/identification_service_areas/...`)
- Any peer USS's endpoints (`/uss/v1/...`)

4.7 Validating Incoming Tokens (Inbound Requests)

Your USS **must validate every inbound request** — whether from DECEA's systems or from other USSs. Failing to do so is a security vulnerability and a homologation failure.

The validation steps are:

Step 1 — Extract the Token

> Extract the JWT from the Authorization header:
>
> Authorization: Bearer <token>

Step 2 — Verify the RS256 Signature

Verify the token's digital signature using **DECEA's public RSA key** (obtained from the Portal UTM during onboarding). The signing algorithm is RS256.

If the signature is invalid → reject with **HTTP 401**.

Step 3 — Verify Token Expiry

Check that the `exp` claim is greater than the current UTC timestamp.

If the token is expired → reject with **HTTP 401**.

Step 4 — Verify the Audience

Check that the `aud` claim matches **your own server's FQDN**. This prevents a token intended for another USS from being replayed against your server.

If `aud` doesn't match your domain → reject with **HTTP 401**.

Step 5 — Verify the Scope

Check that the `scope` claim contains the scope required by the specific endpoint being called. Different endpoints require different scopes (see the OpenAPI specifications for each endpoint's required scope).

If the scope is insufficient → reject with **HTTP 403**.

Go implementation reference (from the BR-UTM workshop):

```
func verifyToken(token string, publicKeyFile string) (bool, error) {
    bytes, _ := os.ReadFile(publicKeyFile)
    publicKey, _ := jwt.ParseRSAPublicKeyFromPEM(bytes)
    parts := strings.Split(token, ".")
    err := jwt.SigningMethodRS256.Verify(
        strings.Join(parts[0:2], "."), parts[2], publicKey,
    )
    return err == nil, err
}
```

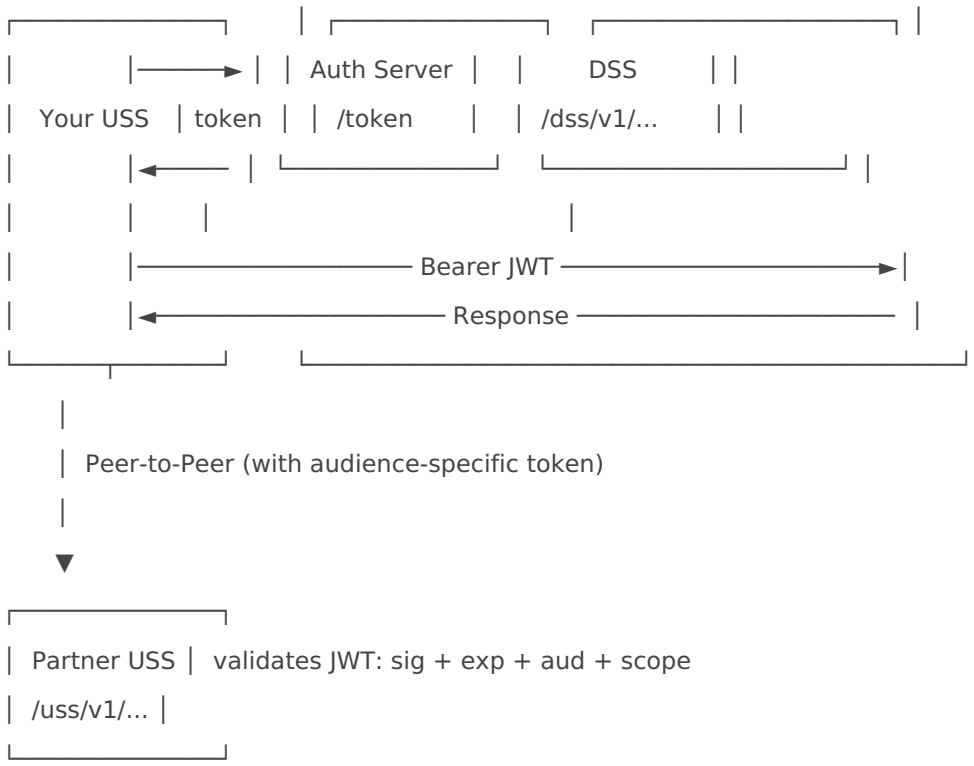
4.8 Token Caching Strategy

For optimal performance:

1. **Cache the token** after obtaining it.
2. **Reuse it** for all requests to the same audience with the same scopes, until it's within a safety margin of expiry (e.g., 60 seconds before `exp`).
3. **Request a new token** when the cached one is near expiry or when you need a different scope/audience combination.

You may maintain multiple cached tokens simultaneously — one per `(scope, audience)` combination that your USS needs.

4.9 Architecture Diagram — Full Authentication Flow



4.10 Security Principles

- **Principle of least privilege**: Request only the scopes your operation actually needs. Avoid requesting all scopes in every token.
- **No token sharing**: Tokens are bound to a specific audience (`aud`). A token obtained to talk to the DSS cannot be used to call another USS, and vice versa.
- **Time synchronization**: Your system clock must be synchronized with DECEA's NTP server (`ntp.decea.gov.br`) to ensure token expiry calculations are accurate. See Chapter 7 — Non-Functional Requirements.

Revision #1
Created 12 July 2026 20:08:13 by João Pedro Favoretti
Updated 14 July 2026 13:19:47 by João Pedro Favoretti