

# OmniHSS Operations Guide

---

## Introduction

OmniHSS is a Home Subscriber Server (HSS) implementation designed for 4G LTE (EPC) and IMS (IP Multimedia Subsystem) networks. As the central database and authentication center for mobile networks, OmniHSS manages subscriber credentials, profile data, and provides authentication and authorization services for both data and voice services.

Built on Elixir and the Erlang VM, OmniHSS delivers high availability, fault tolerance, and scalability required for modern telecommunications infrastructure.

## What is a Home Subscriber Server?

The HSS is a critical component in LTE and IMS networks that:

- **Stores subscriber data** - Credentials, profile information, and service subscriptions
- **Performs authentication** - Validates subscribers attempting to access the network
- **Manages authorization** - Controls which services subscribers can access
- **Tracks location** - Maintains current location information for routing
- **Controls roaming** - Enforces roaming policies based on visited networks
- **Manages equipment** - Functions as Equipment Identity Register (EIR) for device control

# Key Features

## Operational Features

- **S6a Interface** - Authentication and location management for LTE/EPC networks
- **Cx Interface** - IMS registration and authentication
- **Sh Interface** - IMS profile data access and subscription notifications
- **S13 Interface** - Equipment Identity Check (OmniHSS functions as EIR)
- **Gx Interface** - Policy and Charging control (OmniHSS functions as PCRF)
- **Rx Interface** - IMS media policy control (OmniHSS functions as PCRF)
- **Roaming Control** - Granular control over data and IMS roaming by PLMN
- **Multiple MSISDNs** - Support for multiple phone numbers per subscriber
- **RESTful API** - Complete provisioning API for integration (also used by OmniHLR)
- **Web Control Panel** - Real-time monitoring and system status

## Network Element Integration

OmniHSS interfaces with the following network elements:

- **MME** (Mobility Management Entity) - LTE mobility and session management
- **P-GW** (PDN Gateway) - Receives policy from OmniHSS (PCRF function)
- **P-CSCF** (Proxy Call Session Control Function) - IMS media authorization
- **I-CSCF** (Interrogating CSCF) - IMS routing queries
- **S-CSCF** (Serving CSCF) - IMS registration and authentication
- **AS** (Application Server) - IMS subscriber data access
- **OmniHLR** - Legacy HLR that communicates with OmniHSS via API

## Documentation Structure

This operations guide is organized into the following documents:

## Core Documentation

- **Architecture Overview** - System architecture, components, and Diameter stack
- **Configuration Guide** - Complete configuration reference with examples
- **Entity Relationships** - Data model and entity relationships

## Operational Guides

- **Control Panel** - Using the web-based monitoring interface
- **Metrics & Monitoring** - System monitoring and health checks
- **Troubleshooting Guide** - Diagnosing and resolving common issues
- **API Reference** - Complete API endpoint documentation
- **Webhooks** - Real-time event notifications and integration

## Feature Documentation

- **Profile Management** - EPC, IMS, APN, and roaming profiles
- **Roaming Control** - Configuring roaming policies
- **Protocol Flows** - Diameter protocol procedures and message flows
- **PCRF** - Policy and Charging Rules Function (Gx/Rx interfaces, QoS, VoLTE)
- **EIR** - Equipment Identity Register (S13 interface, IMEI validation)
- **Multi-MSISDN and Multi-IMSI Features** - Multiple phone numbers and multiple IMSI support

## High Availability

- **Database Replication** - PostgreSQL logical replication for multi-site HA deployments

# Quick Start for Operations

## Accessing the System

### Control Panel (Web Interface)

```
URL: https://[hostname]:7443
```

The Control Panel provides real-time monitoring of subscribers and Diameter peers.

### API Endpoint

```
URL: https://[hostname]:8443
```

The RESTful API allows provisioning and subscriber management.

## Key Configuration Files

- `config/runtime.exs` - Runtime configuration (database, Diameter, network settings)
- `priv/cert/` - TLS certificates for HTTPS and Diameter

## Essential Operations

1. **Check System Status** - Access Control Panel Overview page
2. **Monitor Diameter Peers** - Access Control Panel Diameter page
3. **Query Subscriber** - Use API endpoint `/api/subscriber/imsi/:imsi`
4. **View Database** - Connect to SQL Database at configured hostname

# Support and Troubleshooting

## Log Files

System logs are output to stdout/stderr and can be captured by your process manager (systemd, supervisord, etc.).

## Common Checks

- **Diameter connectivity** - Check Diameter page for peer status
- **Database connectivity** - Verify database configuration in runtime.exs
- **Subscriber authentication failures** - Check subscriber state for failure counts

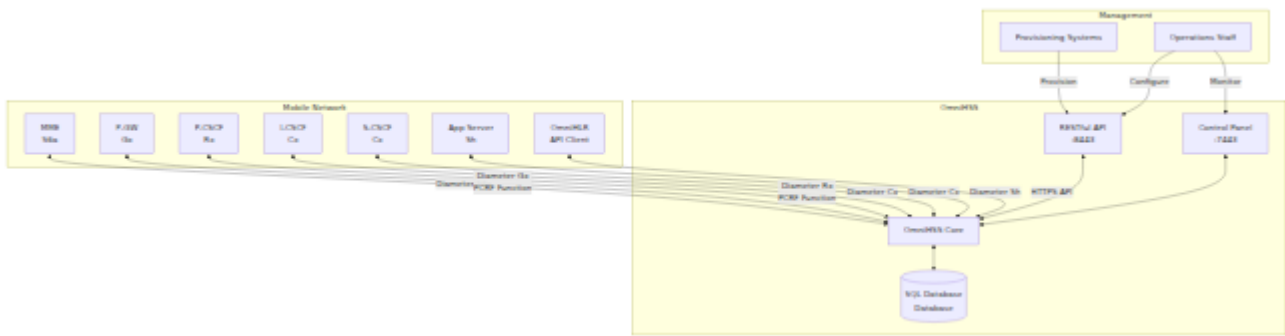
## Health Monitoring

- **API Health Check** - `GET /api/status`
- **Control Panel** - Access any Control Panel page
- **Database** - Connect to SQL Database and verify table access

## Security Considerations

- **TLS Required** - Both API and Control Panel use HTTPS
- **Certificate Management** - Certificates in `priv/cert/` must be valid
- **Database Security** - Secure database credentials in runtime.exs
- **Network Isolation** - Diameter interface should be on management network
- **API Authentication** - Consider implementing authentication for production use

# Architecture at a Glance



## Next Steps

For detailed operational procedures, refer to the specific documentation sections:

- Start with **Architecture Overview** to understand system components
- Review **Configuration Guide** to customize your deployment
- Explore **Control Panel** for day-to-day monitoring
- Consult **API Reference** for provisioning automation

---

**Document Version: 1.0 Maintained By:** Omnitouch Operations Team

# EIR (Equipment Identity Register)

## Overview

The HSS includes a built-in EIR (Equipment Identity Register) that provides equipment identity verification for mobile devices. The EIR validates IMEI (International Mobile Equipment Identity) numbers to determine if mobile equipment is authorized, stolen, or under observation before allowing network access.

## Key Capabilities

- **S13 Interface:** Equipment identity checking via Diameter protocol
- **IMEI Validation:** Verify equipment identity using IMEI/IMEISV
- **Flexible Matching:** Regex-based pattern matching for IMEI, IMEISV, and IMSI
- **Three-Tier Classification:** Whitelist, blacklist, and greylist support
- **Configurable Policies:** Customizable behavior for unknown equipment
- **REST API:** Full CRUD operations for EIR rule management

## Architecture

### Diameter Interface

Interface	Application ID	Peer	Purpose
S13	16,777,252	MME/SGSN	Equipment identity verification

# Equipment Rules Database

The EIR uses a flexible rule-based matching system:

EIR_RULE		
int	id	PK
string	action	
string	regex	
timestamp	inserted_at	
timestamp	updated_at	

## Rule Actions:

- `whitelist` - Allow equipment
- `blacklist` - Block equipment
- `greylist` - Monitor equipment

**Regex Patterns:** Match against IMEI, IMEISV, or IMSI

## Equipment Status Values

Status	Code	Meaning	Network Action
<b>Whitelist</b>	0	Equipment approved	Allow network access
<b>Blacklist</b>	1	Equipment stolen/blocked	Deny network access
<b>Greylist</b>	2	Equipment under observation	Allow with monitoring

# S13 Interface

## Supported Operations

### Equipment Identity Check Request (ECR) / Equipment Identity Check Answer (ECA)

**Direction:** MME/SGSN → HSS (EIR)

**Trigger:** MME verifies equipment identity during attach or tracking area update

#### Request AVPs:

- Session-Id
- Origin-Host, Origin-Realm
- Destination-Realm
- Auth-Session-State
- Terminal-Information
  - IMEI (15 digits)
  - Software-Version (2 digits, optional)
- User-Name (IMSI, optional)
- Vendor-Specific-Application-Id

#### EIR Actions:

1. Extract IMEI, Software-Version (if present), and IMSI (if present)
2. If IMSI provided:
  - Validate subscriber exists and is enabled
  - Update subscriber state with last seen information
3. Attempt equipment lookup in priority order:
  - **IMEISV match** (IMEI + Software-Version concatenated)
  - **IMEI match** (IMEI only)
  - **IMSI match** (if provided in request)
  - **Unknown equipment policy** (configured default behavior)
4. Return equipment status

#### Response AVPs:

- Session-Id (echoed from request)
- Result-Code: 2001 (success)
- Equipment-Status: 0 (whitelist) / 1 (blacklist) / 2 (greylist)

### **Error Responses:**

- Experimental-Result: 5422 (equipment/subscriber not found)
- Experimental-Result: 5012 (general error)

# **Equipment Matching Logic**

## **Priority Order**

The EIR uses a cascading lookup strategy to maximize matching flexibility:

1. IMEISV (IMEI + Software-Version)  
↓ (if no match)
2. IMEI only  
↓ (if no match)
3. IMSI (if provided in request)  
↓ (if no match)
4. Unknown Equipment Policy

## **Matching Algorithm**

### **Step 1: IMEISV Matching**

- Concatenate IMEI + Software-Version: "35979139461611" + "08" = "3597913946161108"
- Test against all EIR rule regex patterns
- Return action ("whitelist", "blacklist", "greylist") of first matching rule

### **Step 2: IMEI Matching (fallback)**

- Use IMEI only: "35979139461611"
- Test against all EIR rule regex patterns

- Return action of first matching rule

### Step 3: IMSI Matching (fallback if IMSI provided)

- Use IMSI from request: "999999876543210"
- Test against all EIR rule regex patterns
- Return action of first matching rule
- **Use case:** Block all equipment for a specific subscriber

### Step 4: Unknown Equipment Policy (final fallback)

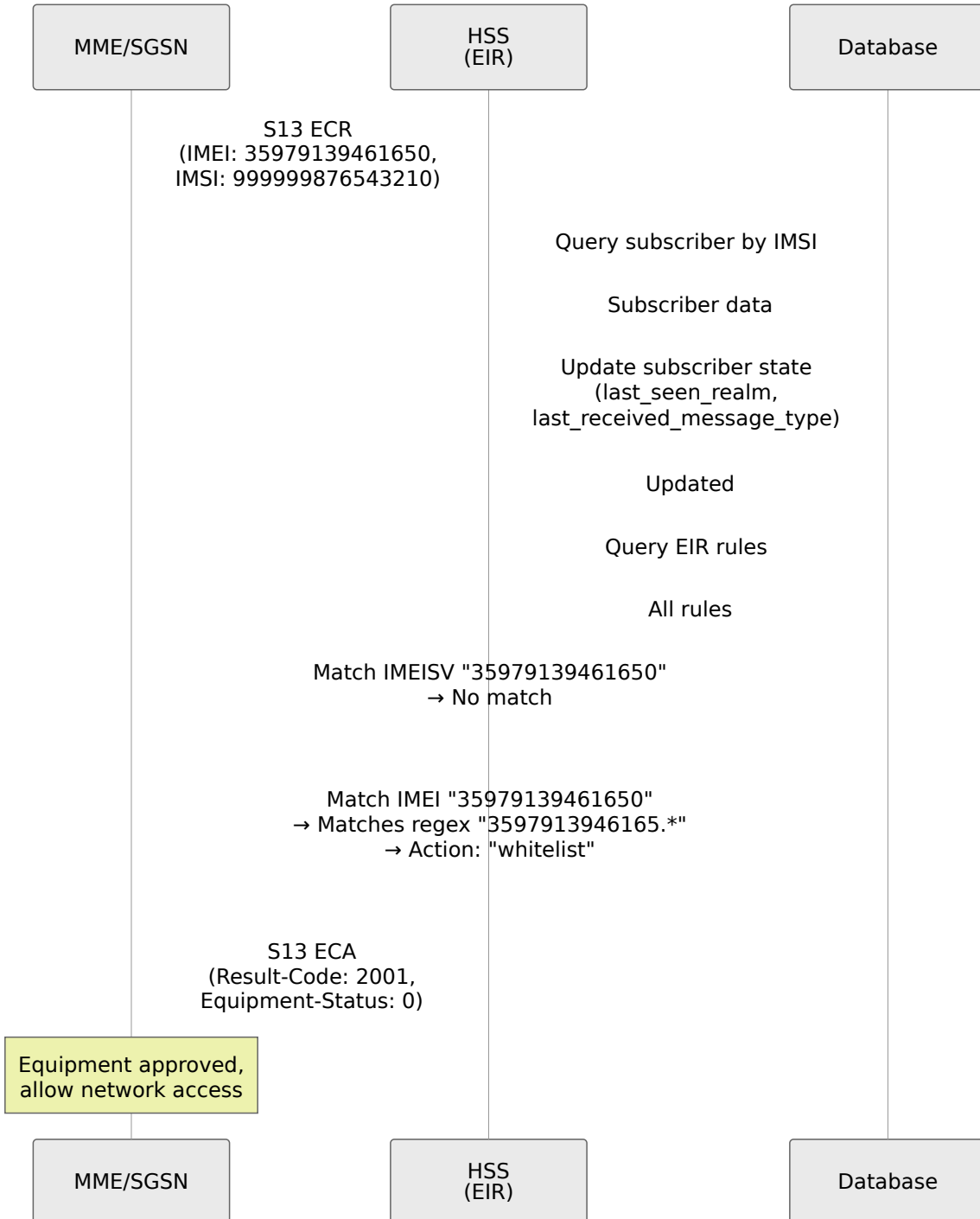
- Configuration setting: eir\_unknown\_equipment\_behaviour
- Options:
  - :whitelist - Allow unknown equipment (permissive)
  - :blacklist - Block unknown equipment (restrictive)
  - :greylist - Observe unknown equipment (moderate)
  - :reject\_unknown\_equipment - Return error 5422 (strict)

## Regex Pattern Examples

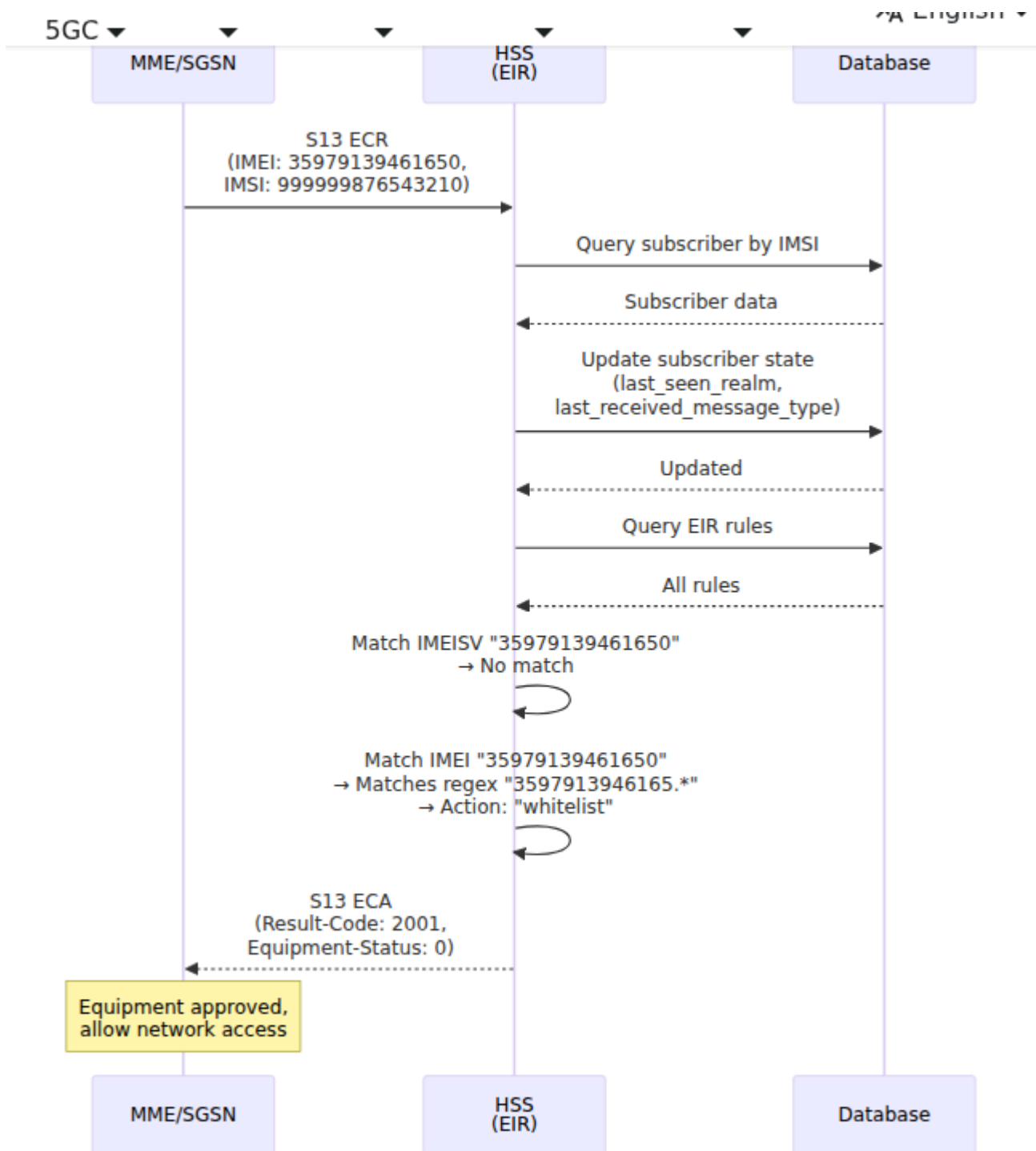
Pattern	Matches	Use Case
"35979139461650"	Exact IMEI	Single device whitelist/blacklist
"3597913946165.*"	IMEI prefix wildcard	Manufacturer/model range
"3597913946161108"	Exact IMEISV	Specific device with software version
"999999876543210"	IMSI	Block all equipment for subscriber
"359791.*"	TAC wildcard	Entire device type allocation

# Common Message Flows

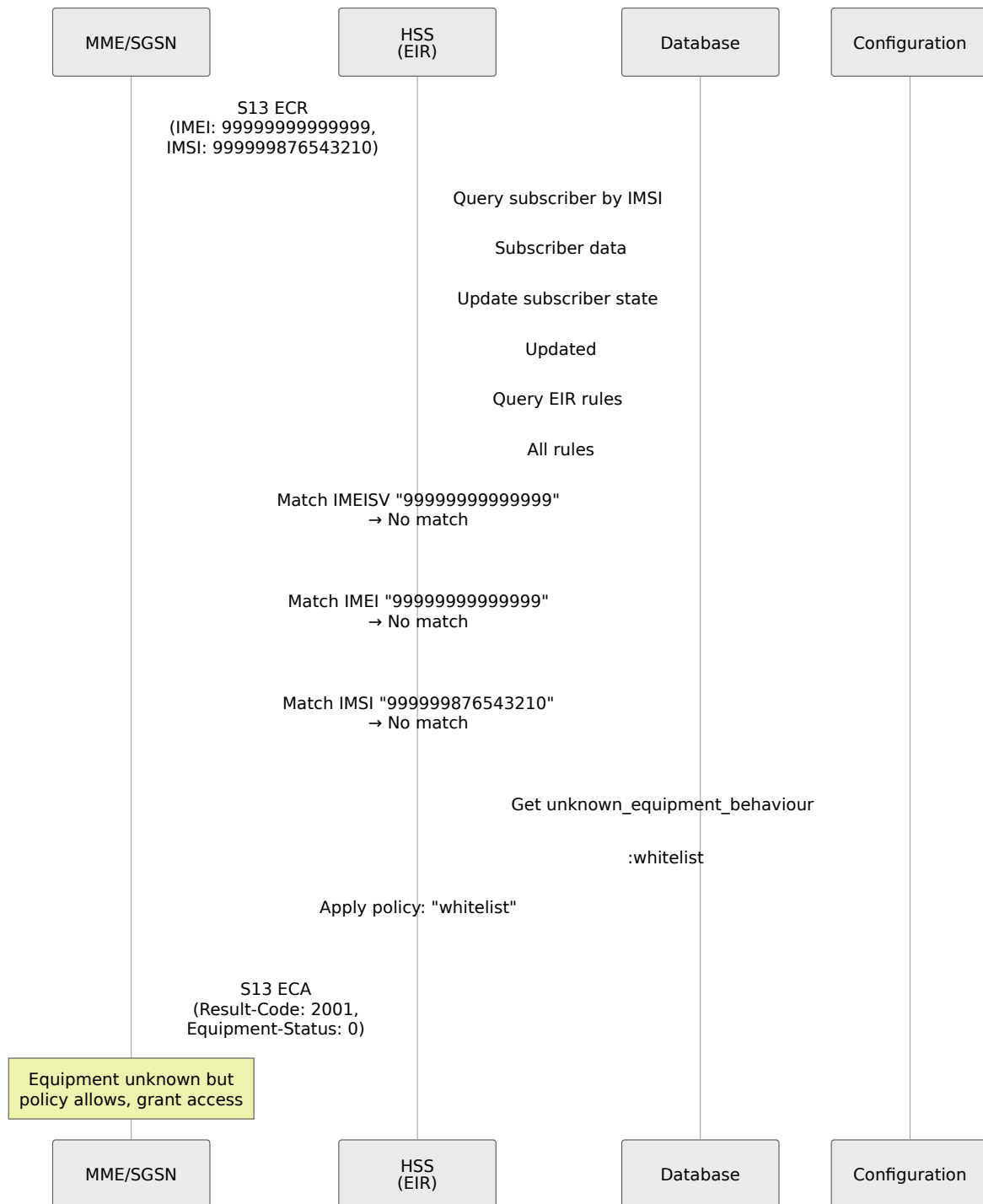
## Flow 1: Equipment Check - Known Whitelisted IMEI



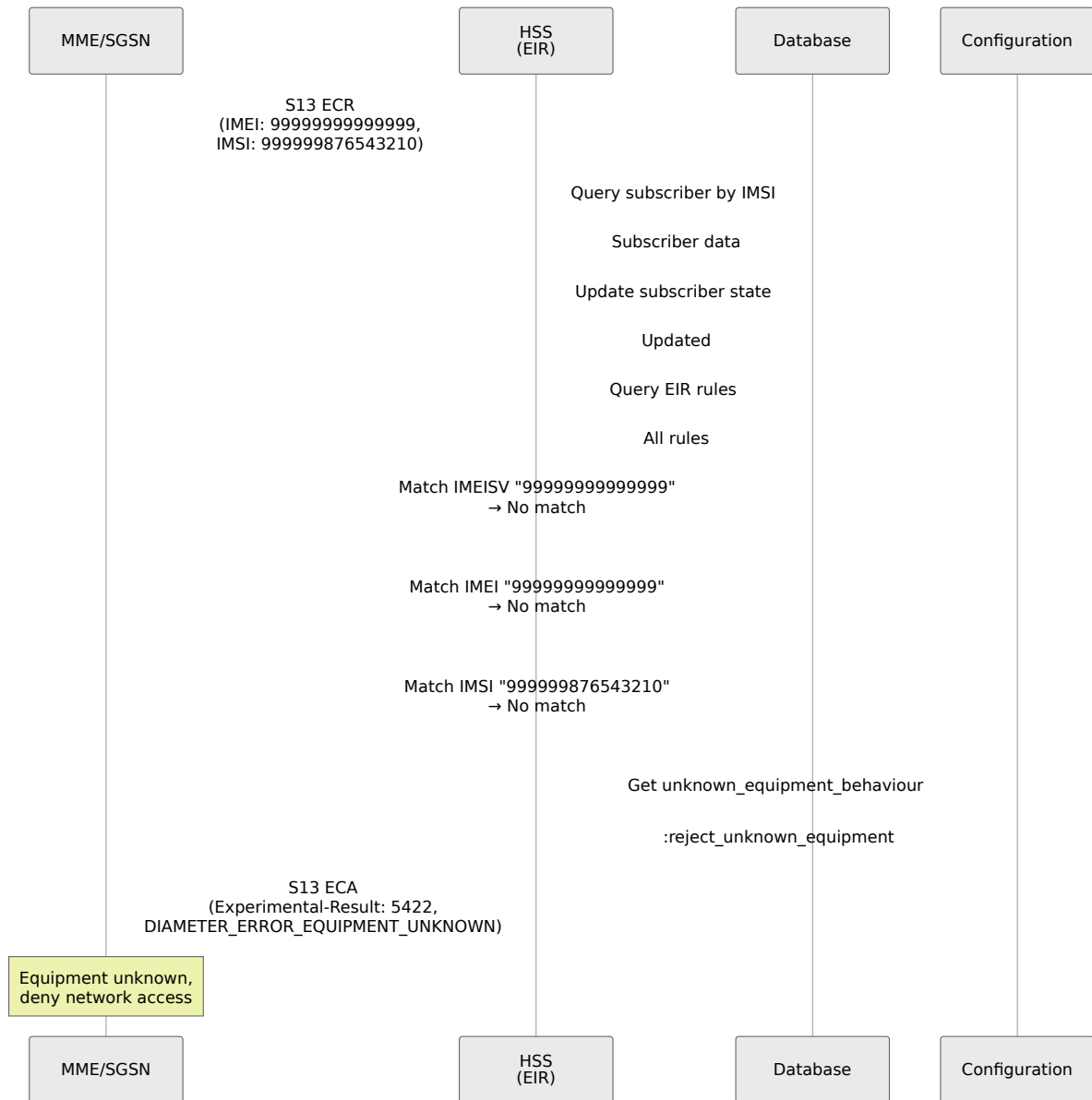
# Flow 2: Equipment Check - Blacklisted IMEI (Stolen Device)



# Flow 3: Equipment Check - Unknown Equipment (Whitelist Policy)



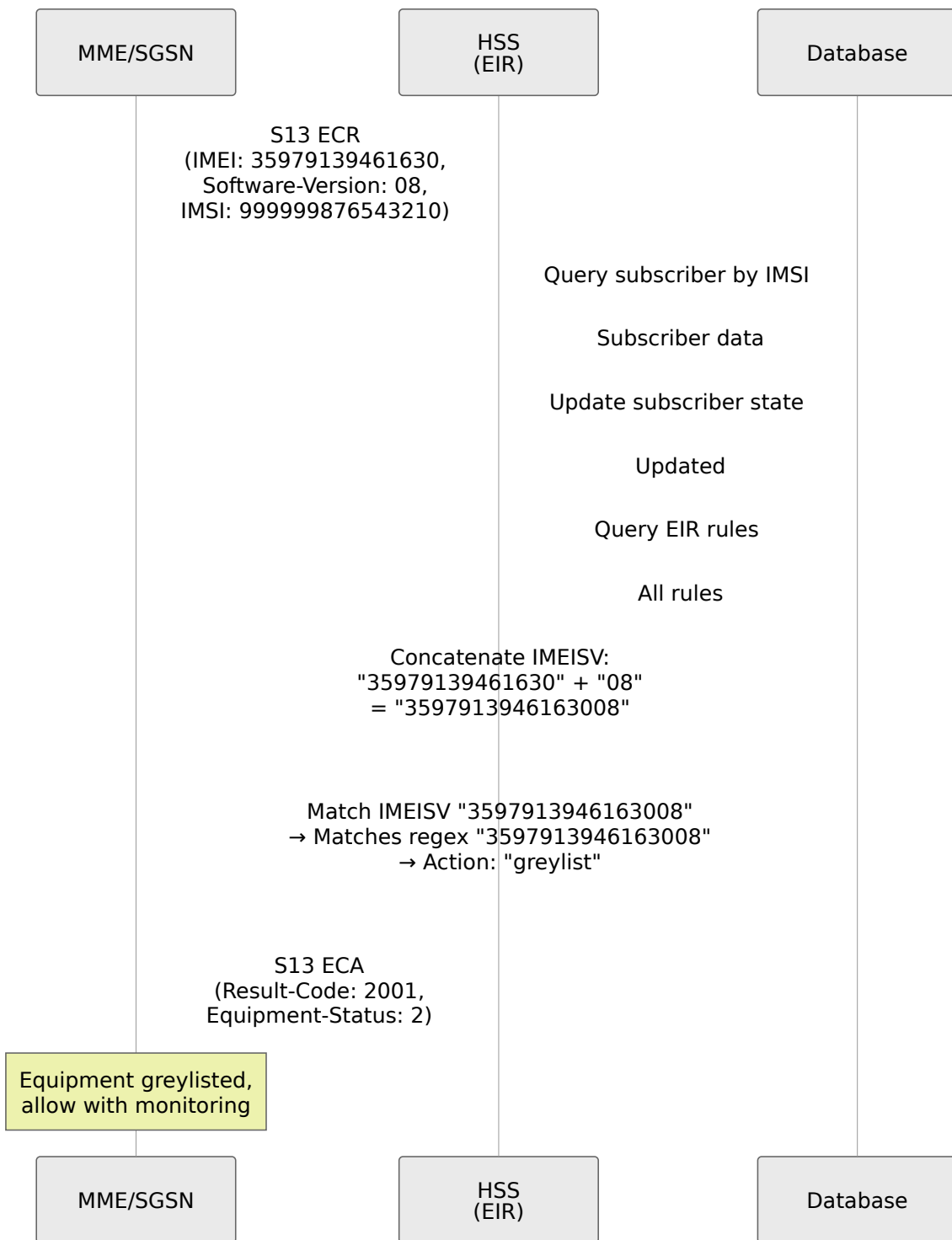
# Flow 4: Equipment Check - Unknown Equipment (Reject Policy)



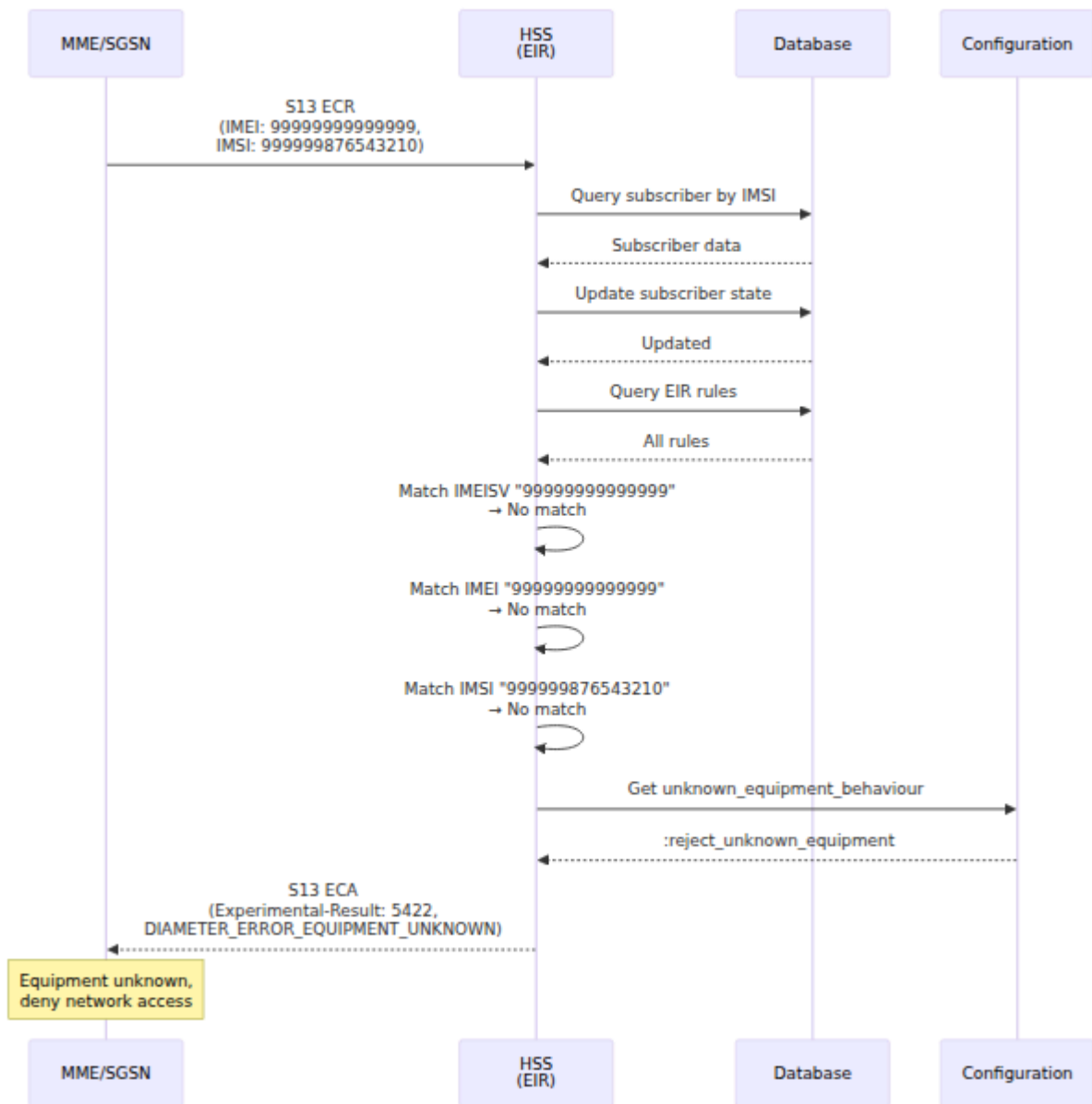
# Flow 5: Equipment Check - Unknown Subscriber



# Flow 6: Equipment Check - IMEISV Match



# Flow 7: Equipment Check - IMSI Blocking



## REST API

### EIR Rule Management

Base path: `/api/eir/rule`

#### List All EIR Rules

**Request:**

```
GET /api/eir/rule
```

**Response** (HTTP 200):

```
{
  "data": [
    {
      "id": 1,
      "action": "whitelist",
      "regex": "3597913946165.*",
      "inserted_at": "2025-01-15T10:30:00Z",
      "updated_at": "2025-01-15T10:30:00Z"
    },
    {
      "id": 2,
      "action": "blacklist",
      "regex": "35979139461640",
      "inserted_at": "2025-01-16T14:20:00Z",
      "updated_at": "2025-01-16T14:20:00Z"
    }
  ]
}
```

**Get Specific EIR Rule**

**Request:**

```
GET /api/eir/rule/{id}
```

**Response** (HTTP 200):

```
{
  "data": {
    "id": 1,
    "action": "whitelist",
    "regex": "3597913946165.*"
  }
}
```

## Create EIR Rule

### Request:

```
POST /api/eir/rule
Content-Type: application/json

{
  "action": "blacklist",
  "regex": "35979139461640"
}
```

### Validation:

- `action`: Required, must be "whitelist", "blacklist", or "greylist"
- `regex`: Required, must be valid regex pattern, unique across all rules

### Response (HTTP 201):

```
{
  "data": {
    "id": 3,
    "action": "blacklist",
    "regex": "35979139461640"
  }
}
```

### Error Response (HTTP 400):

```
{
  "errors": {
    "regex": ["has already been taken"]
  }
}
```

## Update EIR Rule (Partial)

### Request:

```
PATCH /api/eir/rule/{id}
Content-Type: application/json
```

```
{
  "action": "greylist"
}
```

**Response (HTTP 200):**

```
{
  "data": {
    "id": 3,
    "action": "greylist",
    "regex": "35979139461640"
  }
}
```

## Replace EIR Rule

**Request:**

```
PUT /api/eir/rule/{id}
Content-Type: application/json
```

```
{
  "action": "whitelist",
  "regex": "359791394616.*"
}
```

**Response (HTTP 200):**

```
{
  "data": {
    "id": 3,
    "action": "whitelist",
    "regex": "359791394616.*"
  }
}
```

## Delete EIR Rule

### Request:

```
DELETE /api/eir/rule/{id}
```

**Response** (HTTP 204 No Content)

# Configuration

## Diameter Service Setup

### S13 Application (`config/runtime.exs`):

```
%{  
  application_name: :s13,  
  application_dictionary: :diameter_gen_3gpp_s13,  
  vendor_specific_application_ids: [  
    %{vendor_id: 10415, auth_application_id: 16_777_252}  
  ]  
}
```

## Unknown Equipment Behavior

Configure the default behavior for equipment not matching any rules in `config/runtime.exs`:

### Example:

```
config :hss, :eir,  
  unknown_equipment_behaviour: :whitelist
```

### Valid Values:

- `:whitelist` - Allow unknown equipment (default, permissive)
- `:blacklist` - Block unknown equipment (restrictive)

- `:greylis` - Monitor unknown equipment (moderate)
- `:reject_unknown_equipmen` - Return Diameter error 5422 (strict)

### Use Cases:

- **Development/Testing:** `:whitelis` - Allow all devices
- **Production (permissive):** `:whitelis` - Only block known bad devices
- **Production (moderate):** `:greylis` - Log unknown devices for review
- **Production (strict):** `:reject_unknown_equipmen` - Only allow registered devices

## Error Handling

Result Code	Type	Meaning	Cause
2001	Success	DIAMETER_SUCCESS	Equipment check complete
5422	Experimental	DIAMETER_ERROR_EQUIPMENT_UNKNOWN	Subscriber not found or unknown equipment rejected
5012	Experimental	DIAMETER_ERROR_UNKNOWN	Process error

# Use Cases

## 1. Stolen Device Management

**Scenario:** Device reported stolen

**Action:**

```
POST /api/eir/rule
{
  "action": "blacklist",
  "regex": "35979139461640" # Exact IMEI
}
```

**Result:** Device denied network access on next attachment

## 2. Manufacturer Whitelist

**Scenario:** Pre-approve entire device model range

**Action:**

```
POST /api/eir/rule
{
  "action": "whitelist",
  "regex": "359791394.*" # TAC for manufacturer/model
}
```

**Result:** All devices in TAC range approved

## 3. Subscriber Equipment Lock

**Scenario:** Block all equipment for specific subscriber (SIM lock)

**Action:**

```
POST /api/eir/rule
{
  "action": "blacklist",
  "regex": "999999876543210" # IMSI
}
```

**Result:** Any equipment used with this SIM is blocked

## 4. Test Equipment Greylist

**Scenario:** Monitor test equipment in production

**Action:**

```
POST /api/eir/rule
{
  "action": "greylist",
  "regex": "35979139.*" # Test equipment TAC range
}
```

**Result:** Equipment allowed but flagged for monitoring

## 5. Software Version Control

**Scenario:** Block specific vulnerable firmware version

**Action:**

```
POST /api/eir/rule
{
  "action": "blacklist",
  "regex": "359791394616.*05" # IMEI range + Software Version 05
}
```

**Result:** Only devices with Software-Version "05" in IMEI range blocked

# Implementation Details

## Internal Components

The EIR functionality is implemented using several internal modules:

- **S13 Protocol Handler** - ECR/ECA message processing
- **Equipment Matching Engine** - Regex-based IMEI/IMEISV/IMSI matching
- **EIR Rules Database** - Pattern storage and lookup
- **REST API Controller** - Rule management endpoints

## Equipment Status Lookup Function

The equipment status lookup follows this cascading logic:

1. **IMEISV Matching**: Check IMEI + Software-Version concatenated
2. **IMEI Matching**: Check IMEI only
3. **IMSI Matching**: Check IMSI (if provided)
4. **Unknown Equipment**: Apply configured default policy

### Possible Results:

- `whitelist` - Equipment allowed
- `blacklist` - Equipment blocked
- `greylist` - Equipment under observation
- `reject_unknown_equipment` - Strict rejection

# Security Considerations

## IMEI Privacy

IMEI numbers are sensitive identifiers. The EIR:

- Does not log IMEI values in plaintext by default
- Uses hashed database lookups (if configured)

- Restricts API access to authenticated administrators

## Rule Ordering

EIR rules are evaluated in database order (by ID). For conflicting patterns:

```
Rule 1: regex "359791.*" action "whitelist" (broad)
Rule 2: regex "35979139461640" action "blacklist" (specific)
```

**Recommendation:** Create specific rules before broad wildcards to ensure blacklist takes precedence.

## Rate Limiting

Consider implementing rate limiting on:

- S13 ECR requests from untrusted peers
- REST API EIR rule modifications
- IMEI lookup queries to prevent enumeration attacks

## Related Documentation

- [Diameter Protocols](#) - S13 protocol specification
- [API Reference](#) - Complete API documentation
- [Architecture](#) - Overall HSS architecture
- [Operations Guide](#) - Operational procedures

# Appendix: IMEI Structure

## IMEI Format (15 digits)

```
35 9791 394616 1
| | |      └─ Check digit (Luhn algorithm)
| | └───┬─ Serial Number (6 digits)
| └───┬─ FAC (Final Assembly Code, 4 digits)
└───┬─ TAC (Type Allocation Code, 8 digits total including RBI)
    | └─ RBI (Reporting Body Identifier, 2 digits)
    └─ Manufacturer/Model (6 digits)
```

## IMEISV Format (16 digits)

```
35 9791 394616 1 08
| | | | | └─ Software Version (2 digits)
└───┬─ IMEI (15 digits)
```

## Example Patterns

IMEI/IMEISV	Pattern	Matches
359791394616108	3597913946161.*	All devices with TAC+FAC+Serial 359791394616*
359791394616140	35979139461614.	All check digits for Serial 359791394616141-9
35979139461640	35979139461640	Exact IMEI match
3597913946163008	3597913946163008	Exact IMEISV (IMEI + SV) match

# PCRF (Policy and Charging Rules Function)

## Overview

The HSS includes a built-in PCRF (Policy and Charging Rules Function) that provides policy control and charging rules for mobile data sessions. The PCRF controls Quality of Service (QoS) policies, bandwidth allocation, and charging rules for both default and dedicated bearers in LTE networks.

## Key Capabilities

- **Gx Interface:** Policy control for PGW/PCEF (Packet Data Network Gateway / Policy and Charging Enforcement Function)
- **Rx Interface:** Authorization and QoS for IMS (IP Multimedia Subsystem) media flows
- **Dynamic Policy Management:** Real-time policy updates via Re-Auth Requests (RAR)
- **VoLTE Support:** Dedicated bearer creation for voice calls with guaranteed QoS
- **Charging Rules:** Define charging behavior and speed profiles using Traffic Flow Templates (TFTs)
- **REST API:** Programmatic control of policy enforcement and rule management

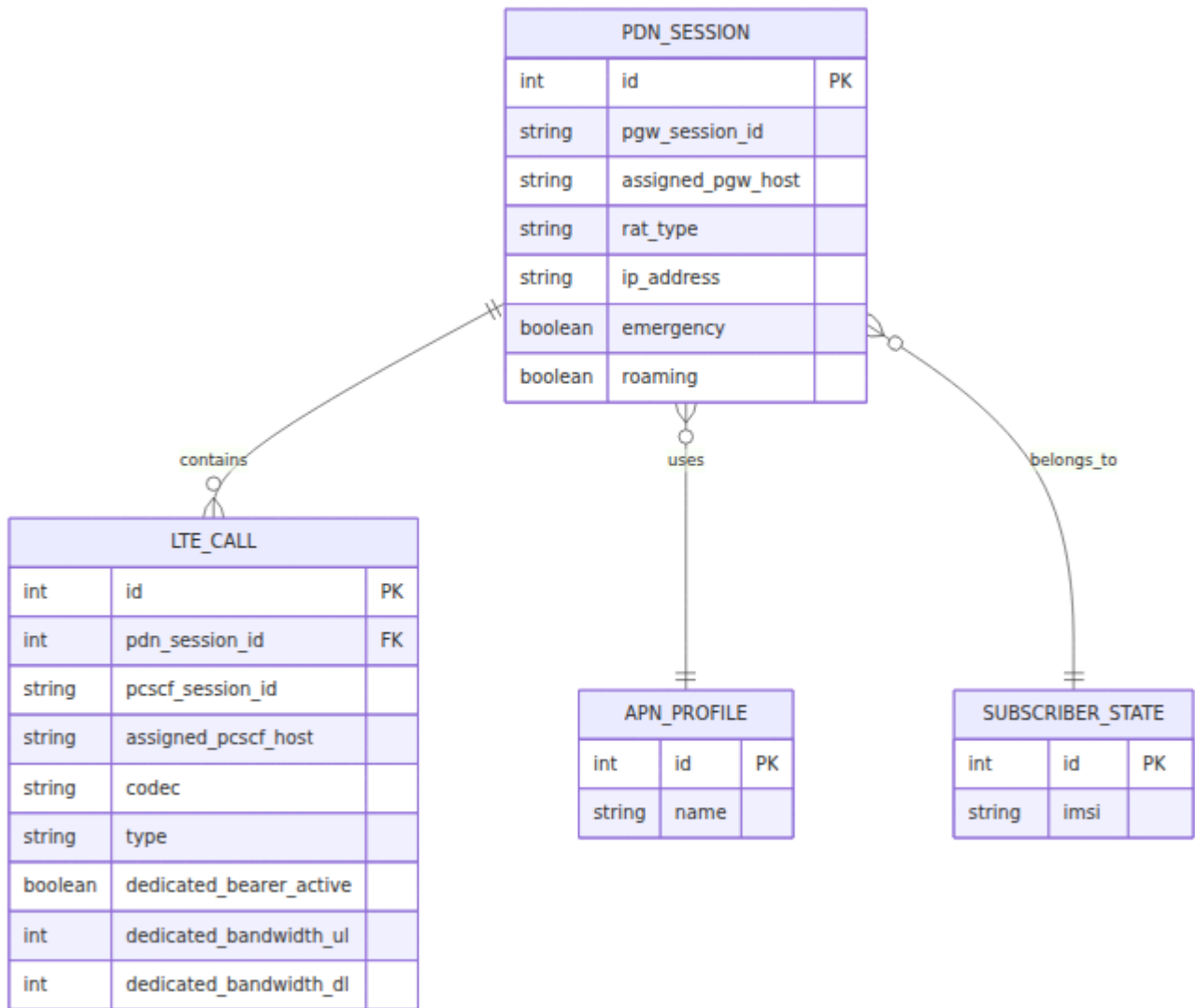
# Architecture

## Diameter Interfaces

Interface	Application ID	Peer	Purpose
<b>Gx</b>	16,777,238	PGW (PCEF)	PDN session management, QoS enforcement, charging rules
<b>Rx</b>	16,777,236	P-CSCF (AF)	IMS media authorization, bandwidth reservation

## Session State Management

The PCRF maintains session state for active PDN connections and VoLTE calls:



# Gx Interface

## Supported Operations

### 1. Credit Control Request - Initial (CCR-I)

**Trigger:** PGW creates new PDN connection for subscriber

#### Request AVPs:

- Session-Id
- Origin-Host, Origin-Realm
- Subscription-Id (contains IMSI)
- Called-Station-Id (APN name)

- IP-CAN-Type (IP Connectivity Access Network type)
- RAT-Type (Radio Access Technology)
- Framed-IP-Address (UE IP address)

#### **PCRF Actions:**

1. Lookup subscriber by IMSI
2. Retrieve APN profile and QoS configuration
3. Create session tracking entry
4. Build QoS policies from APN profile

#### **Response AVPs:**

- Result-Code: 2001 (DIAMETER\_SUCCESS)
- QoS-Information (APN aggregate bandwidth limits)
- Default-EPS-Bearer-QoS (QCI, ARP, priority)
- Bearer-Control-Mode
- Online / Offline (session-level default charging method — see [Session-Level Online/Offline Charging](#))

## **2. Credit Control Request - Update (CCR-U)**

**Trigger:** PGW reports session changes (location update, RAT change, etc.)

#### **PCRF Actions:**

1. Locate existing session by session ID
2. Update session parameters (RAT type, location, etc.)
3. Return updated policies if needed

**Response:** Result-Code 2001 with optional policy updates

## **3. Credit Control Request - Terminate (CCR-T)**

**Trigger:** PGW terminates PDN connection

#### **PCRF Actions:**

1. Locate session by session ID

2. Delete session and associated call records
3. Confirm termination

**Response:** Result-Code 2001

## Session-Level Online/Offline Charging

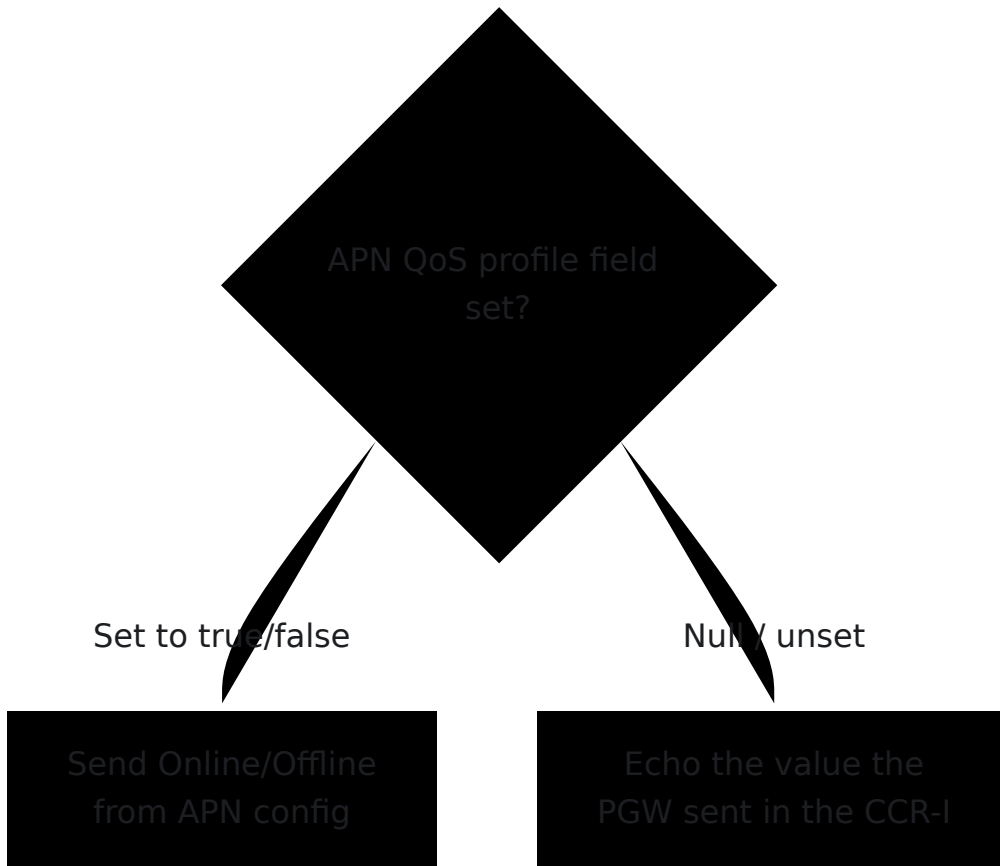
The `Online` (AVP 1009) and `Offline` (AVP 1008) AVPs sent at the top level of a Gx CCA-I set the **default charging method for the whole IP-CAN session** (the default bearer), as defined in [3GPP TS 29.212](#). This is distinct from the per-rule `Online/Offline` carried *inside* a Charging-Rule-Definition, which only governs the traffic matched by that specific rule.

By default the PCRF **echoes** whichever values the PGW/PCEF advertised in the CCR-I. The APN QoS profile can instead **pin** the session-level charging method, overriding the echo. This is controlled by two optional fields on the APN QoS profile:

Field	Type	AVP	Effect
<code>online_charging_enabled</code>	Boolean	<code>Online</code> (1009)	<code>true</code> → <code>ENABLE_ONLINE (1)</code> , <code>false</code> → <code>DISABLE_ONLINE (0)</code>
<code>offline_charging_enabled</code>	Boolean	<code>Offline</code> (1008)	<code>true</code> → <code>ENABLE_OFFLINE (1)</code> , <code>false</code> → <code>DISABLE_OFFLINE</code> <code>(0)</code>

Each field is resolved independently:

Build Gx CCA-I



**Charging Method AVP Values (TS 29.212):**

AVP	Code	Value	Name	Meaning
Online	1009	0	DISABLE_ONLINE	Online charging (OCS / Gy) not applied to the session
Online	1009	1	ENABLE_ONLINE	Online charging applied — credit must be granted before traffic flows
Offline	1008	0	DISABLE_OFFLINE	Offline charging (CDR / Gz) not applied to the session
Offline	1008	1	ENABLE_OFFLINE	Offline charging applied — usage recorded for billing

**How it works:** When `online_charging_enabled`/`offline_charging_enabled` are left unset (the default for every existing profile), behaviour is unchanged — the PCRF mirrors the PGW's request, so the network's existing charging posture is preserved. Setting either field forces that charging mode on every CCR-I for any APN using the profile, regardless of what the PGW advertises.

**Use case:** Pin online charging on for a prepaid APN (so the PGW must consult the OCS even if it requested offline), or force offline charging on a postpaid/IoT APN to guarantee CDR generation. Because the setting lives on the APN QoS profile, it applies uniformly to every APN that references that profile.

**Note:** This controls the *session default* only. Per-traffic charging is still governed by the `online_charging_enabled`/`offline_charging_enabled` fields on individual **Charging Rules**, which appear inside Charging-Rule-Definition AVPs.

#### 4. Re-Auth Request (RAR)

**Direction:** PCRF → PGW (HSS initiates)

## **Trigger:**

- IMS call setup (Rx AAR triggers Gx RAR)
- IMS call teardown (Rx STR triggers Gx RAR)
- Manual re-auth via REST API

## **RAR AVPs:**

- Session-Id (PGW session ID)
- Auth-Application-Id: 16,777,238
- Re-Auth-Request-Type (0 = Authorize only)
- Charging-Rule-Install/Remove
- QoS-Information (for dedicated bearers)

**PGW Actions:** Create/modify/delete dedicated bearers based on charging rules

# **Charging Rules and Traffic Flow Templates**

The PCRF supports defining charging rules with Traffic Flow Templates (TFTs) to control:

- **Service-specific charging** - Different rates for video, gaming, social media, etc.
- **Speed profiles** - Throttle or prioritize traffic matching specific patterns
- **Usage-based policies** - Apply different QoS based on traffic type or destination

Charging rules can be:

- Installed dynamically via Gx RAR based on application detection
- Pre-defined and triggered by specific conditions (time of day, location, quota)
- Associated with TFTs using packet filter rules (5-tuple: protocol, source/dest IP, source/dest port)

## **Common Use Cases:**

- **Zero-rating** - Unlimited access to specific services (Spotify, WhatsApp, Facebook) without consuming data quota
- **Post-quota access** - Allow self-care portal and support sites even after subscriber exhausts data allowance
- **Tiered speed** - High-speed for premium services, throttled for standard content
- **Time-based policies** - Off-peak unlimited streaming, peak-time prioritization
- **Roaming policies** - Different charging for international vs domestic data usage
- **Enterprise SLAs** - Guaranteed QoS for business-critical applications

## QoS Policy Structure

**Default Bearer QoS** (from APN profile):

```
{
  "QoS-Class-Identifier": 9,           // QCI (9 = default bearer)
  "APN-Aggregate-Max-Bitrate-UL": 50000, // kbps
  "APN-Aggregate-Max-Bitrate-DL": 100000, // kbps
  "Allocation-Retention-Priority": {
    "Priority-Level": 8,
    "Pre-emption-Capability": 1,       // May preempt
    "Pre-emption-Vulnerability": 1    // May be preempted
  }
}
```

**Dedicated Bearer QoS** (for VoLTE):

```
{
  "QoS-Class-Identifier": 1,           // QCI 1 = Conversational
  Voice
  "Max-Requested-Bandwidth-UL": 128000, // bps
  "Max-Requested-Bandwidth-DL": 128000, // bps
  "Guaranteed-Bitrate-UL": 128000,
  "Guaranteed-Bitrate-DL": 128000
}
```

# Rx Interface

## Supported Operations

### 1. AA Request (AAR) / AA Answer (AAA)

**Trigger:** P-CSCF requests authorization for IMS media session (VoLTE call setup)

#### Request AVPs:

- Session-Id (P-CSCF session identifier)
- Subscription-Id (IMSI or SIP URI)
- Media-Component-Description
  - Media-Type (audio, video)
  - Max-Requested-Bandwidth-UL/DL
  - Codec-Data
  - Flow-Description (5-tuple packet filters)
- AF-Application-Identifier

#### PCRF Actions:

1. Lookup subscriber by IMSI or SIP URI
2. Find active IMS session
3. Extract media parameters (codec, bandwidth, flow rules)
4. Create call tracking entry
5. **Trigger Gx RAR to PGW** to create dedicated bearer
6. Wait for Gx RAA response
7. Return Rx AAA with authorization result

#### Response AVPs:

- Result-Code: 2001 (success) or 5063 (service not authorized)

#### Emergency (SOS) calls, including unauthenticated / SIM-less callers

Emergency calls (`urn:service:sos`) are authorized even when the caller is not a known subscriber (a SIM-less device, or a roaming subscriber with no

credentials on file). This is required so that emergency calls always obtain the correct QoS (TS 23.167).

How the PCRF recognises and authorizes an emergency AAR:

1. **Emergency detection.** The request is treated as emergency if **either** the `Subscription-Id` is a SIP URI containing `sos`, **or** the AAR's `Framed-IP-Address` resolves to a PDN session that was already created as an emergency PDN by the Gx CCR-I (the Gx side flags emergency by the APN / `Called-Station-Id` containing `sos`). The second path is what lets an **anonymous** emergency INVITE (no usable identity) still be authorized.
2. **Subscriber resolution.** Looked up by `Subscription-Id`, else by `Framed-IP-Address`. An emergency **orphan PDN** (created by Gx for an unknown subscriber) has no subscriber (`subscriber_state_id: nil`); the AAR then takes the orphan path rather than the normal-subscriber path.
3. **SOS APN profile.** The dedicated-bearer QoS is taken from the SOS APN profile (see "Built-in default SOS APN profile" below).
4. A Gx RAR installs the dedicated voice bearer (QCI 1) on the (orphan) PDN, and the AAA returns `2001`.

Note: a `Framed-IP-Address` only yields authorization if it maps to a PDN that Gx already created **as emergency**. A non-emergency request from an unregistered UE is still rejected at the P-CSCF (it never reaches the PCRF as an emergency AAR).

**Built-in default SOS APN profile.** Emergency authorization no longer depends on an operator having pre-provisioned a `sos` APN profile in the database. The PCRF resolves the SOS APN profile as:

- an operator-defined APN profile whose APN identifier is `sos`, if one exists; otherwise
- a **built-in default** (`Hss.Control.Common.default_sos_apn_profile/0`).

The built-in default is a transient profile (no DB row, `apn_profile_id: nil` on the orphan PDN — the column is nullable) with emergency QoS:

Parameter	Value
Default SOS bearer QCI	5
Dedicated voice bearer QCI	1 (ARP priority 2)
Allocation-Retention-Priority	1
Pre-emption-Capability	enabled
Pre-emption-Vulnerability	disabled
APN-AMBR UL/DL	10000 kbps

To override these values (or to set per-operator charging rules via a PCRF profile), create a `sos` APN profile in the DB — it takes precedence over the built-in default.

## 2. Session Termination Request (STR) / Session Termination Answer (STA)

**Trigger:** P-CSCF terminates IMS session (call hangup)

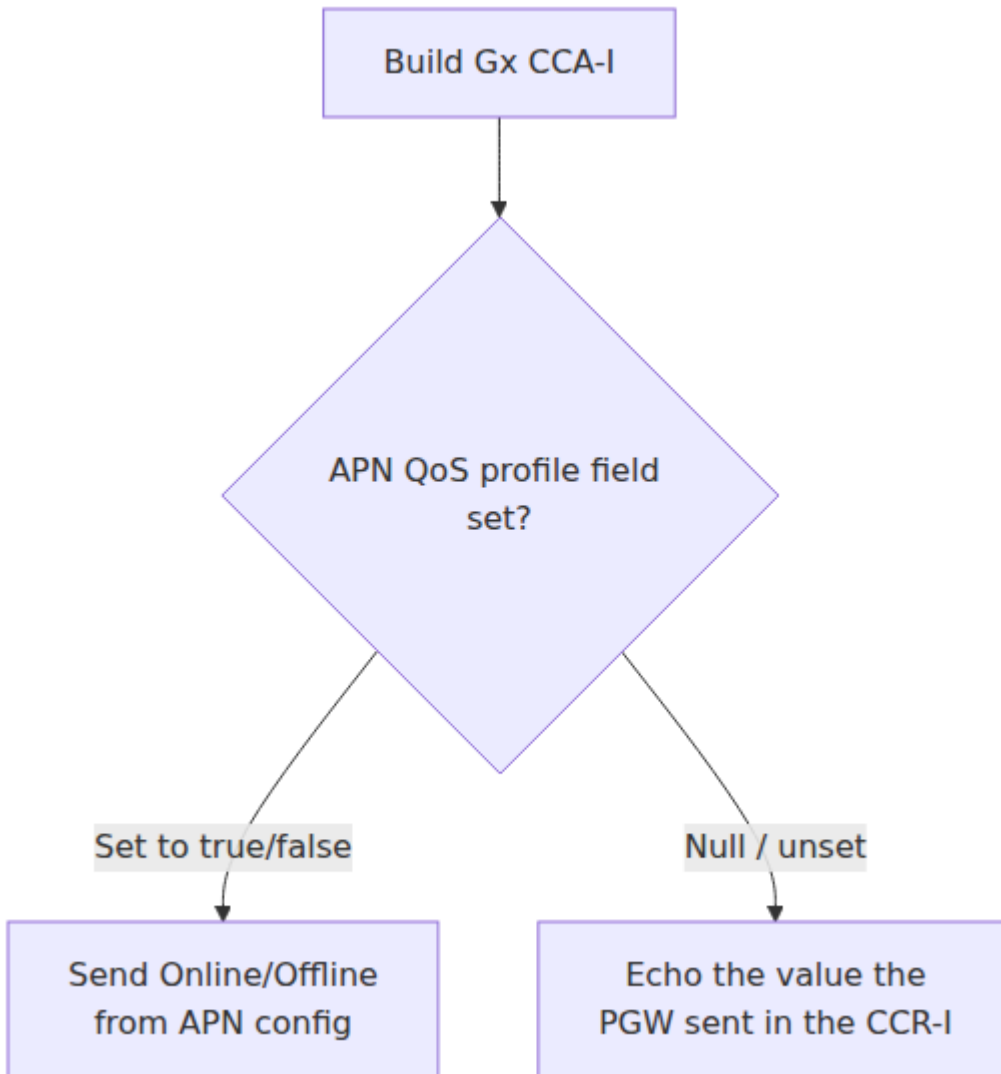
### PCRF Actions:

1. Locate call session by P-CSCF session ID
2. **Trigger Gx RAR to PGW** to remove dedicated bearer
3. Delete call tracking entry
4. Return STA confirmation

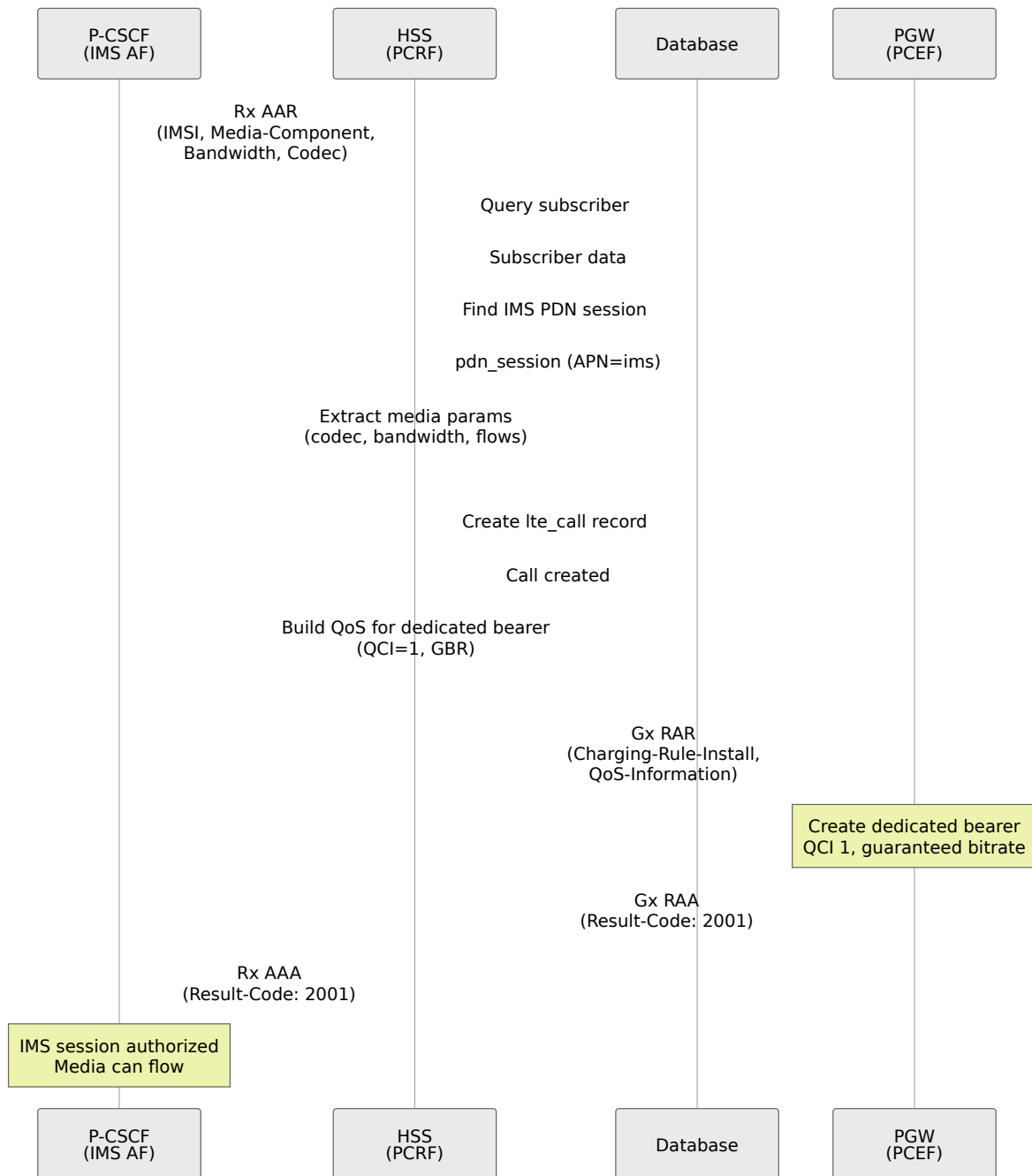
**Response:** Result-Code 2001

# Common Message Flows

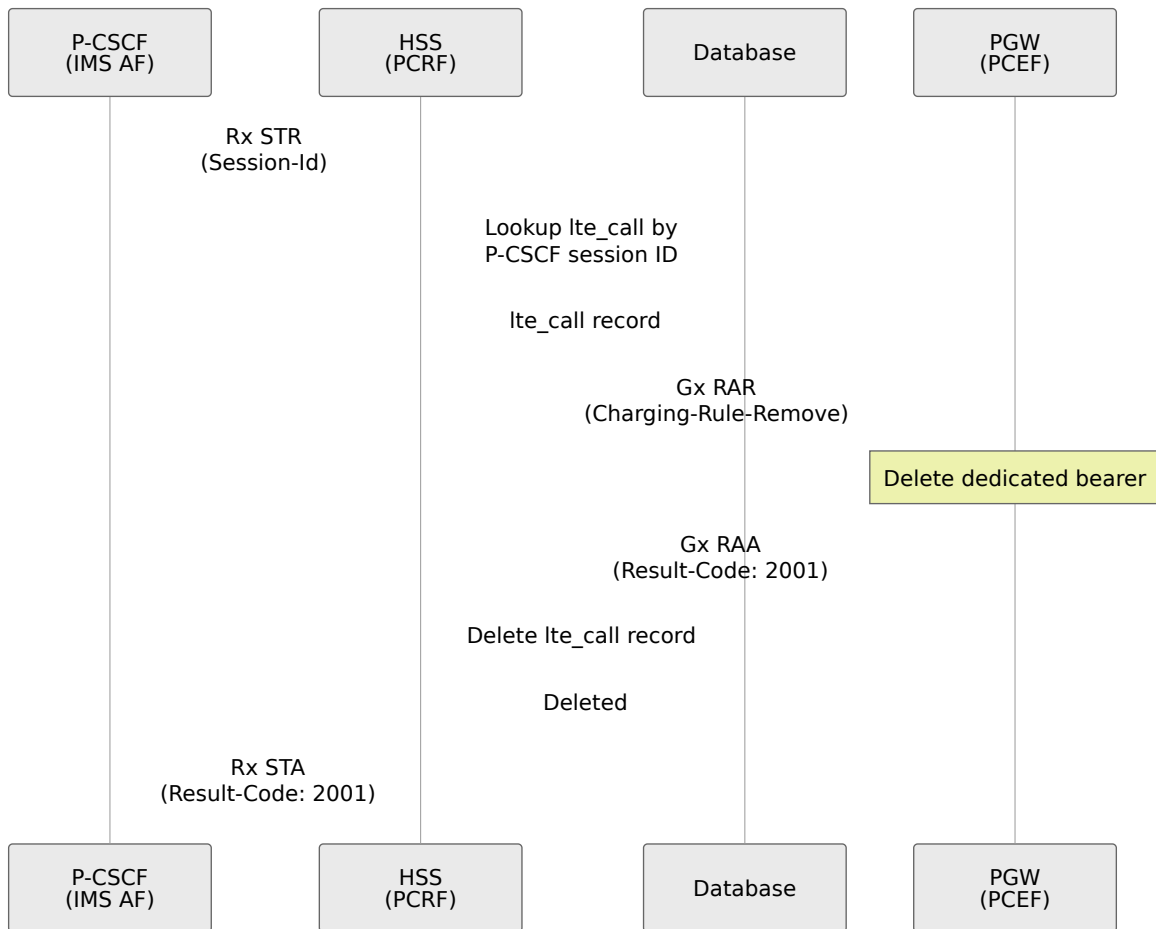
## Flow 1: PDN Session Establishment



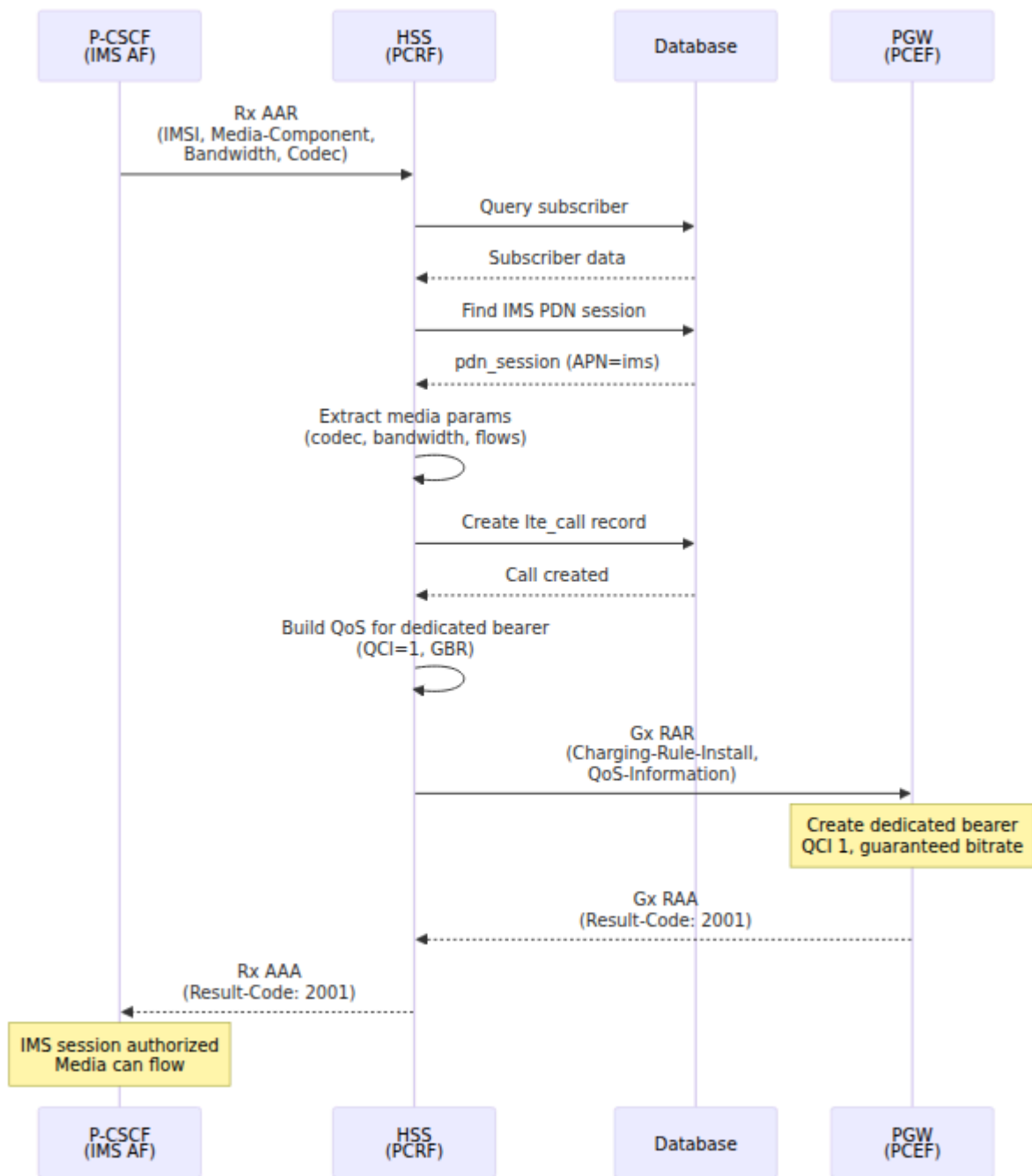
# Flow 2: VoLTE Call Setup (Rx AAR → Gx RAR)



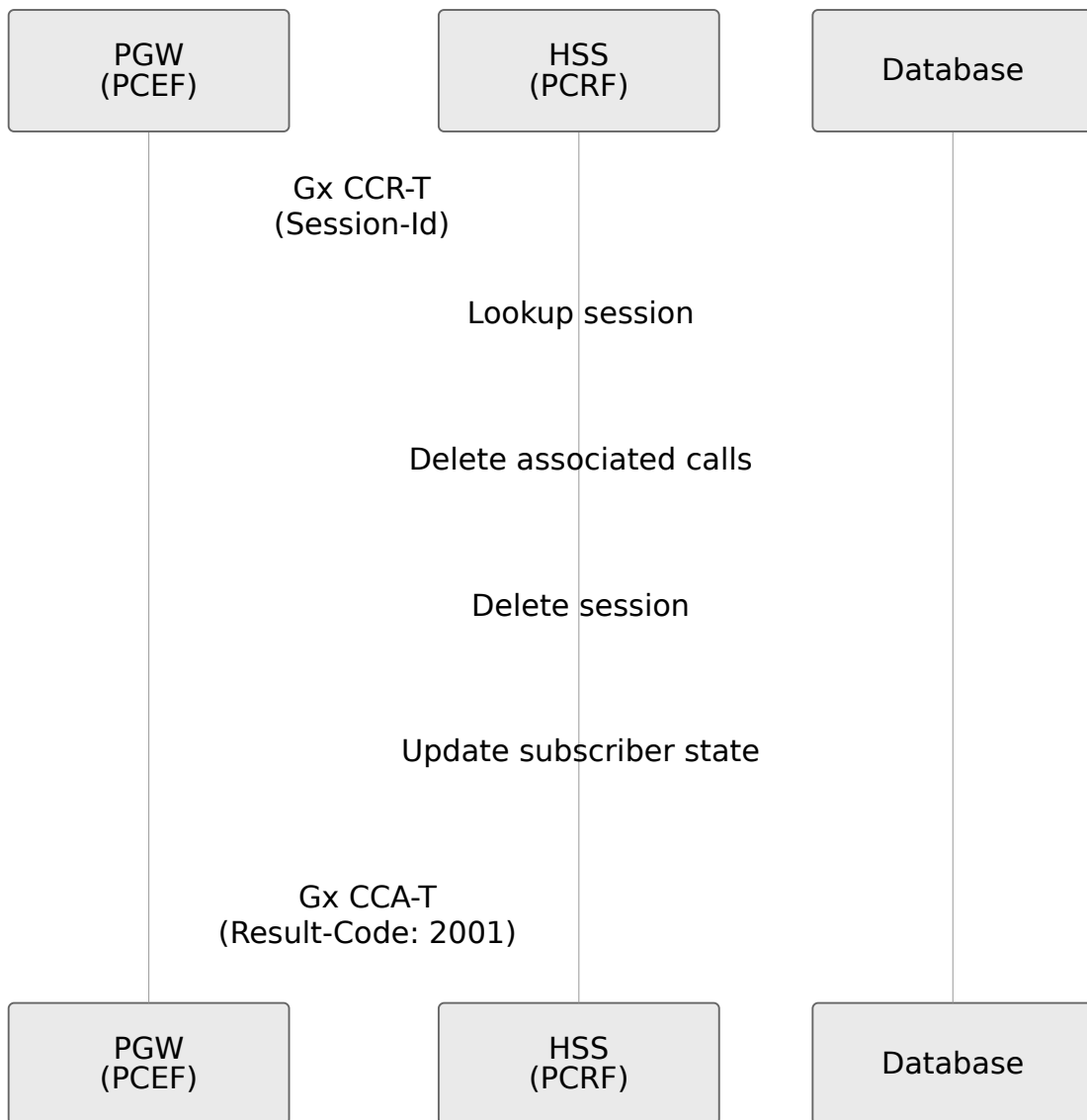
# Flow 3: VoLTE Call Teardown (Rx STR → Gx RAR)



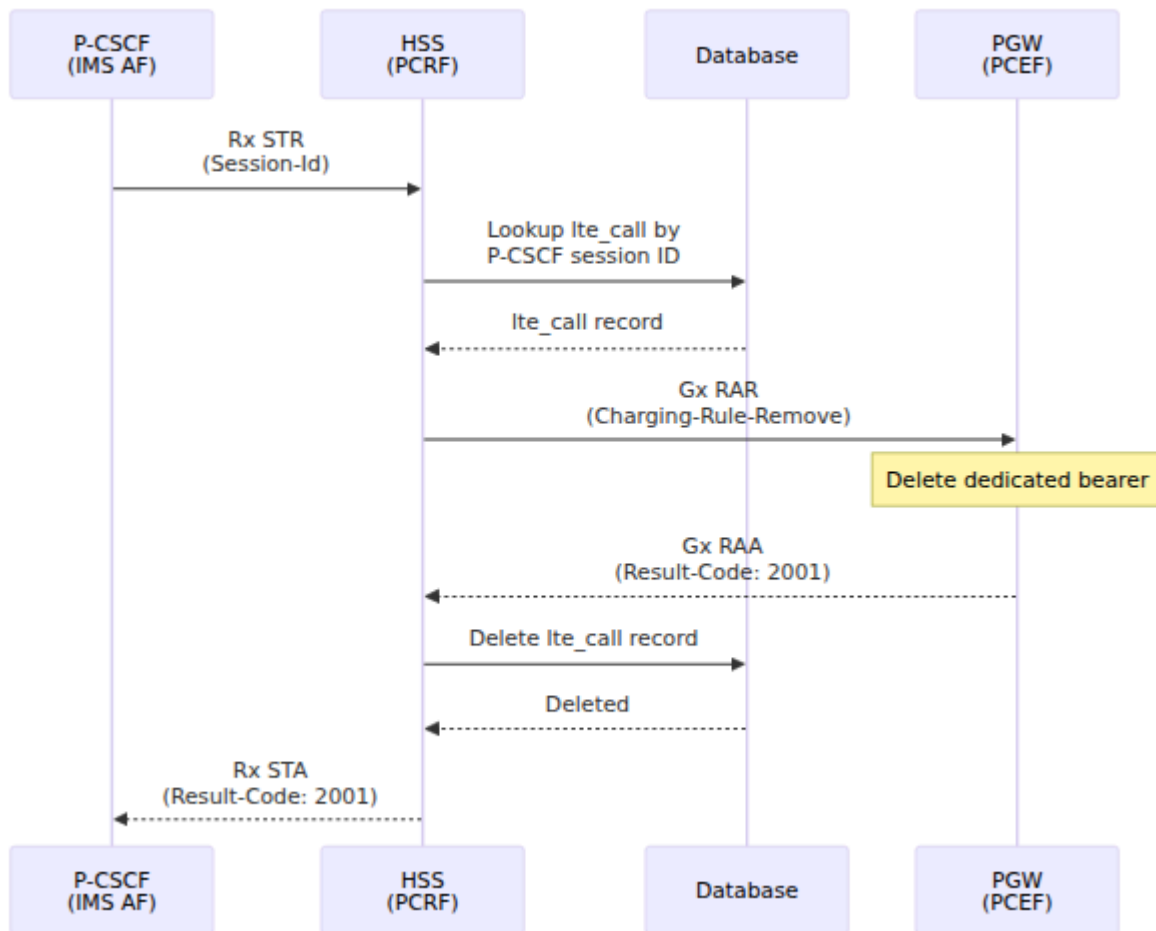
# Flow 4: PDN Session Update



# Flow 5: PDN Session Termination



## Flow 6: Manual Re-Auth via REST API



## REST API

### PCRF Re-Auth Endpoint

**Endpoint:** `POST /api/operation/pcrf_re_auth`

**Purpose:** Manually trigger Gx Re-Auth Request to refresh policies

**When to Use:** This manual endpoint is typically used for troubleshooting or forcing policy refresh on specific subscribers. For routine policy updates (changing APN QoS profiles), the system automatically triggers re-auth for all affected sessions - no manual action needed.

**Request Body:**

```
{
  "imsi": "999999876543210",
  "apn": "ims"
}
```

**Success Response** (HTTP 200):

```
{
  "data": "Gx Re-Auth Request for 999999876543210 sent to
pgw.epc.mnc999.mcc999.3gppnetwork.org, Result-Code: 2001"
}
```

**Error Response** (HTTP 400):

```
{
  "error": "Unable to send Re-Auth Request for 999999876543210 on
APN ims, no active PDN Session found"
}
```

## Policy Configuration API

The PCRF retrieves QoS policies from APN configurations stored in the database. These policies can be created and managed via REST API.

**Automatic Policy Enforcement:** When you update an APN QoS profile (e.g., change bandwidth limits or QCI), the system automatically sends Gx Re-Auth Requests (RAR) to all PGWs with active PDN sessions using that APN. This ensures policy changes are applied immediately to all connected subscribers without manual intervention.

### Policy Architecture

Policies are defined through a three-tier structure:

APN Identifier → APN QoS Profile → APN Profile

↓

"internet"

↓

QCI, AMBR, ARP

↓

Links both together

## 1. Create APN Identifier

Define the APN name and IP version support.

**Endpoint:** POST /api/apn/identifier

### Request Body:

```
{
  "apn_identifier": {
    "apn": "internet",
    "ip_version": "ipv4v6"
  }
}
```

### IP Version Options:

- "ipv4" - IPv4 only
- "ipv6" - IPv6 only
- "ipv4v6" - Dual stack (both IPv4 and IPv6)
- "ipv4\_or\_ipv6" - Network decides (either IPv4 or IPv6)

### Response (HTTP 201):

```
{
  "data": {
    "id": 1,
    "apn": "internet",
    "ip_version": "ipv4v6"
  }
}
```

### Validation:

- `apn`: Required, 1-254 characters, unique
- `ip_version`: Required, must be one of the four options above

**List APN Identifiers:** `GET /api/apn/identifier`

## 2. Create APN QoS Profile

Define the QoS parameters (bandwidth, QCI, priority).

**Endpoint:** `POST /api/apn/qos_profile`

### Request Body:

```
{
  "apn_qos_profile": {
    "name": "Best Effort Internet",
    "qci": 9,
    "allocation_retention_priority": 8,
    "apn_ambr_dl_kbps": 100000,
    "apn_ambr_ul_kbps": 50000,
    "pre_emption_capability": false,
    "pre_emption_vulnerability": true,
    "online_charging_enabled": false,
    "offline_charging_enabled": true
  }
}
```

### QoS Parameters:

Field	Type	Range	Description
name	string	1-254 chars	Profile name (unique)
qci	integer	1-254	QoS Class Identifier (1-4 = GBR, 5-9 = Non-GBR)
allocation_retention_priority	integer	1-15	ARP level (1 = highest priority)
apn_ambr_dl_kbps	integer	1-4,294,967,293	APN Aggregate Maximum Bit Rate Downlink (kbps)
apn_ambr_ul_kbps	integer	1-4,294,967,293	APN Aggregate Maximum Bit Rate Uplink (kbps)
pre_emption_capability	boolean	true/false	Can preempt lower priority bearers
pre_emption_vulnerability	boolean	true/false	Can be preempted by higher priority bearers

Field	Type	Range	Description
online_charging_enabled	boolean	true/false/null	Session-level <b>Online</b> AVP for the Gx CCA-I. <b>null</b> (default) echoes the PGW's CCR-I value. See <a href="#">Session-Level Online/Offline Charging</a>
offline_charging_enabled	boolean	true/false/null	Session-level <b>Offline</b> AVP for the Gx CCA-I. <b>null</b> (default) echoes the PGW's CCR-I value. See <a href="#">Session-Level Online/Offline Charging</a>

### Common QCI Values:

- **1** - Conversational Voice (VoLTE) - GBR, 100ms delay budget
- **2** - Conversational Video - GBR, 150ms delay budget
- **5** - IMS Signaling - Non-GBR, 100ms delay budget
- **9** - Default Bearer (Internet) - Non-GBR, 300ms delay budget

### Response (HTTP 201):

```
{
  "data": {
    "id": 1,
    "name": "Best Effort Internet",
    "qci": 9,
    "allocation_retention_priority": 8,
    "apn_ambr_dl_kbps": 100000,
    "apn_ambr_ul_kbps": 50000,
    "pre_emption_capability": false,
    "pre_emption_vulnerability": true,
    "online_charging_enabled": false,
    "offline_charging_enabled": true
  }
}
```

**List QoS Profiles:** GET /api/apn/qos\_profile

### 3. Create APN Profile

Link the APN identifier with a QoS profile.

**Endpoint:** POST /api/apn/profile

#### Request Body:

```
{
  "apn_profile": {
    "name": "Internet APN",
    "apn_identifier_id": 1,
    "apn_qos_profile_id": 1
  }
}
```

#### Fields:

- `name`: Profile name (unique), used for reference
- `apn_identifier_id`: ID from [Create APN Identifier](#)
- `apn_qos_profile_id`: ID from [Create APN QoS Profile](#)

**Response (HTTP 201):**

```
{
  "data": {
    "id": 1,
    "name": "Internet APN",
    "apn_identifier_id": 1,
    "apn_qos_profile_id": 1
  }
}
```

### Constraints:

- `apn_identifier_id` and `apn_qos_profile_id` must reference existing records
- Each combination of APN identifier and QoS profile must be unique

**List APN Profiles:** `GET /api/apn/profile`

### Complete Policy Configuration Example

#### Step 1: Create IMS APN Policy (VoLTE)

```
# 1. Create APN Identifier
curl -X POST https://hss.example.com:8443/api/apn/identifier \
-H "Content-Type: application/json" \
-d '{
  "apn_identifier": {
    "apn": "ims",
    "ip_version": "ipv4v6"
  }
}'
# Response: {"data": {"id": 2, ...}}

# 2. Create QoS Profile (IMS Signaling)
curl -X POST https://hss.example.com:8443/api/apn/qos_profile \
-H "Content-Type: application/json" \
-d '{
  "apn_qos_profile": {
    "name": "IMS Signaling QoS",
    "qci": 5,
    "allocation_retention_priority": 2,
    "apn_ambr_dl_kbps": 5000,
    "apn_ambr_ul_kbps": 5000,
    "pre_emption_capability": true,
    "pre_emption_vulnerability": false
  }
}'
# Response: {"data": {"id": 2, ...}}

# 3. Create APN Profile
curl -X POST https://hss.example.com:8443/api/apn/profile \
-H "Content-Type: application/json" \
-d '{
  "apn_profile": {
    "name": "IMS APN",
    "apn_identifier_id": 2,
    "apn_qos_profile_id": 2
  }
}'
# Response: {"data": {"id": 2, ...}}
```

## Step 2: Assign to Subscriber

Once created, the APN profile is assigned to subscribers via EPC profiles. See [API Reference](#) for linking APN profiles to subscribers.

## Policy Update and Deletion

### Update QoS Profile:

```
PATCH /api/apn/qos_profile/{id}
PUT /api/apn/qos_profile/{id}
```

### Example - Increase Bandwidth for All Users:

```
# Update QoS profile ID 1 to increase bandwidth
curl -X PATCH https://hss.example.com:8443/api/apn/qos_profile/1 \
-H "Content-Type: application/json" \
-d '{
  "apn_qos_profile": {
    "apn_ambr_dl_kbps": 150000,
    "apn_ambr_ul_kbps": 75000
  }
}'
```

### What Happens Automatically:

1. QoS profile is updated in the database
2. System identifies all active PDN sessions using APNs linked to this QoS profile
3. For each active session, a Gx RAR is sent to the corresponding PGW
4. PGWs update bearer QoS to reflect new bandwidth limits
5. All connected subscribers immediately receive the updated policy

**Example Scenario:** If 100 subscribers are currently connected on the "internet" APN using QoS profile ID 1, all 100 will have their bandwidth limits updated to 150 Mbps down / 75 Mbps up within seconds of the API call completing.

**Note:** When you update an APN QoS profile, the system **automatically triggers re-auth** for all active PDN sessions using that APN, applying the new

policies immediately to attached subscribers. No manual re-auth is required.

### Delete Resources:

```
DELETE /api/apn/identifier/{id}
DELETE /api/apn/qos_profile/{id}
DELETE /api/apn/profile/{id}
```

### Deletion Constraints:

- Cannot delete APN identifiers or QoS profiles referenced by APN profiles
- Cannot delete APN profiles assigned to active subscribers

### Policy Templates

#### High-Speed Internet (100 Mbps down / 50 Mbps up):

```
{
  "apn_qos_profile": {
    "name": "High Speed Internet",
    "qci": 9,
    "allocation_retention_priority": 8,
    "apn_ambr_dl_kbps": 100000,
    "apn_ambr_ul_kbps": 50000,
    "pre_emption_capability": false,
    "pre_emption_vulnerability": true
  }
}
```

#### Premium Internet (500 Mbps down / 100 Mbps up):

```
{
  "apn_qos_profile": {
    "name": "Premium Internet",
    "qci": 8,
    "allocation_retention_priority": 5,
    "apn_ambr_dl_kbps": 500000,
    "apn_ambr_ul_kbps": 100000,
    "pre_emption_capability": true,
    "pre_emption_vulnerability": false
  }
}
```

### IoT/M2M (Low Bandwidth):

```
{
  "apn_qos_profile": {
    "name": "IoT M2M",
    "qci": 9,
    "allocation_retention_priority": 10,
    "apn_ambr_dl_kbps": 1024,
    "apn_ambr_ul_kbps": 512,
    "pre_emption_capability": false,
    "pre_emption_vulnerability": true
  }
}
```

### Emergency Services (Highest Priority):

```
{
  "apn_qos_profile": {
    "name": "Emergency APN",
    "qci": 5,
    "allocation_retention_priority": 1,
    "apn_ambr_dl_kbps": 10000,
    "apn_ambr_ul_kbps": 10000,
    "pre_emption_capability": true,
    "pre_emption_vulnerability": false
  }
}
```

# Configuration

## Diameter Service Setup

**Gx Application** (`config/runtime.exs`):

```
%{
  application_name: :gx,
  application_dictionary: :diameter_gen_3gpp_gx,
  vendor_specific_application_ids: [
    %{vendor_id: 10415, auth_application_id: 16_777_238}
  ]
}
```

**Rx Application** (`config/runtime.exs`):

```
%{
  application_name: :rx,
  application_dictionary: :diameter_gen_3gpp_rx,
  vendor_specific_application_ids: [
    %{vendor_id: 10415, auth_application_id: 16_777_236}
  ]
}
```

## QoS Parameters

QoS parameters are sourced from:

- **Default Bearer:** APN profile configuration in database
  - `apn_qos_profile.qci` (QoS Class Identifier)
  - `apn_qos_profile.apn_ambr_ul_kbps` (Aggregate Maximum Bit Rate Uplink)
  - `apn_qos_profile.apn_ambr_dl_kbps` (Aggregate Maximum Bit Rate Downlink)
  - `apn_qos_profile.priority_level` (Allocation Retention Priority)

- **Dedicated Bearer:** Extracted from Rx AAR Media-Component-Description
  - QCI: 1 (Conversational Voice)
  - Guaranteed Bitrate: From Max-Requested-Bandwidth AVPs
  - Flow filters: From Flow-Description AVPs

## Error Handling

Result Code	Type	Meaning	Cause
2001	Success	DIAMETER_SUCCESS	Request processed successfully
5001	Experimental	User not found	IMSI not in subscriber database
5002	Experimental	Session not found	PDN session doesn't exist for update/terminate
5063	Experimental	Service not authorized	IMS media authorization denied

## Implementation Details

### Session Management

The PCRF tracks:

- **Active PDN Sessions** - One per APN, per subscriber
- **VoLTE Calls** - Multiple calls per IMS session (supports conference calling)
- **QoS Policies** - Applied dynamically based on APN configuration
- **Charging Rules** - Traffic flow templates and service-specific policies

# Advanced Policy Features

The PCRF supports advanced policy control including:

- **Charging rule installation/removal** via Gx interface
- **Traffic Flow Template (TFT) matching** for service differentiation
- **Dynamic speed profiles** based on application or traffic type
- **Service-aware policies** triggered by network conditions or subscriber behavior

Contact your system administrator for information on configuring advanced charging rules and TFT-based policies.

## Related Documentation

- [Diameter Protocols](#) - Detailed protocol specifications
- [API Reference](#) - Complete API documentation
- [Architecture](#) - Overall HSS architecture
- [Data Mapping](#) - Database to Diameter AVP mappings

# API Error Handling

[← Back to API Reference](#)

---

## Table of Contents

- [Common Error Responses](#)
  - [Error Handling Flow](#)
- 

## Common Error Responses

### 400 Bad Request

```
{  
  "error": "Invalid JSON format"  
}
```

#### Causes:

- Malformed JSON
- Missing required fields
- Invalid data types

### 404 Not Found

```
{  
  "error": "Resource not found"  
}
```

#### Causes:

- Subscriber/profile/entity doesn't exist
- Incorrect ID in URL

## 422 Unprocessable Entity

```
{
  "errors": {
    "imsi": ["has already been taken"],
    "key_set_id": ["does not exist"]
  }
}
```

### Causes:

- Validation failures
- Database constraints violated
- Foreign key references don't exist

## 500 Internal Server Error

```
{
  "error": "Internal server error"
}
```

### Causes:

- Database connectivity issues
  - Unexpected application errors
-

# Error Handling Flow

API Request

OmniCore  
5GC

OmniCall

OmniRAN

OmniCharge

Platform

English

Invalid JSON

Valid

400 Bad Request

Authorized?

No

Yes

401 Unauthorized

Resource Exists?

No

Yes

404 Not Found

Data Valid?

No

Yes

422 Validation Error

Process Request

Database OK?

Error

Success

500 Server Error

200/201 Success

---

[← Back to API Reference](#)

# API Usage Examples

[← Back to API Reference](#)

---

## Table of Contents

- [Complete Subscriber Provisioning](#)
  - [Complete Static IP Provisioning](#)
- 

## Complete Subscriber Provisioning

This example demonstrates the complete workflow for provisioning a new subscriber from scratch. The process involves creating all required profiles and components before creating the subscriber.

**Prerequisites:** This example uses `jq` for JSON parsing. Install with `apt-get install jq` or `brew install jq`.

### Related Sections:

- [Key Set Management](#)
- [APN Profiles](#)
- [EPC Profiles](#)
- [Subscriber Management](#)

### # 1. Create Key Set

```
KEY_SET_ID=$(curl -k -X POST
https://hss.example.com:8443/api/key_set \
-H "Content-Type: application/json" \
-d '{
  "ki": "0123456789ABCDEF0123456789ABCDEF",
  "opc": "FEDCBA9876543210FEDCBA9876543210",
  "authentication_algorithm": "milenage",
  "amf": "8000",
  "sqn": 0
}' | jq -r '.response.id')
```

### # 2. Create APN QoS Profile

```
APN_QOS_ID=$(curl -k -X POST
https://hss.example.com:8443/api/apn/qos_profile \
-H "Content-Type: application/json" \
-d '{
  "name": "Default Internet QoS",
  "allocation_retention_priority": 8,
  "apn_ambr_dl_kbps": 50000,
  "apn_ambr_ul_kbps": 25000,
  "pre_emption_capability": true,
  "pre_emption_vulnerability": true,
  "qci": 9
}' | jq -r '.response.id')
```

### # 3. Create APN Identifier

```
APN_ID=$(curl -k -X POST
https://hss.example.com:8443/api/apn/identifier \
-H "Content-Type: application/json" \
-d '{
  "apn": "internet",
  "ip_version": "ipv4v6"
}' | jq -r '.response.id')
```

### # 4. Create APN Profile

```
APN_PROFILE_ID=$(curl -k -X POST
https://hss.example.com:8443/api/apn/profile \
-H "Content-Type: application/json" \
-d '{
  "apn_identifier_id": $APN_ID,
  "apn_qos_profile_id": $APN_QOS_ID,
  "name": "Internet APN"
}' | jq -r '.response.id')
```

```

}" | jq -r '.response.id')

# 5. Create EPC Profile
EPC_PROFILE_ID=$(curl -k -X POST
https://hss.example.com:8443/api/epc/profile \
-H "Content-Type: application/json" \
-d "{
  \"apn_profiles\": [\$APN_PROFILE_ID],
  \"name\": \"Standard Data Plan\",
  \"network_access_mode\": \"packet_only\",
  \"tracking_area_update_interval_seconds\": 600,
  \"ue_ambr_dl_kbps\": 100000,
  \"ue_ambr_ul_kbps\": 50000
}" | jq -r '.response.id')

# 6. Create Subscriber
SUBSCRIBER_ID=$(curl -k -X POST
https://hss.example.com:8443/api/subscriber \
-H "Content-Type: application/json" \
-d "{
  \"imsi\": \"001001123456789\",
  \"key_set_id\": $KEY_SET_ID,
  \"epc_profile_id\": $EPC_PROFILE_ID
}" | jq -r '.response.id')

echo "Subscriber provisioned successfully with ID: $SUBSCRIBER_ID"

```

## What This Creates:

This provisioning workflow creates a complete subscriber with:

1. **Cryptographic keys** ([Key Set](#)) - For authentication
2. **Data service profile** ([EPC Profile](#)) - Bandwidth and network access settings
3. **APN configuration** ([APN Profile](#)) - Access point with QoS
4. **Subscriber record** ([Subscriber](#)) - The actual subscriber entity

## Next Steps:

- Add phone numbers: See [MSISDN Management](#)
- Enable voice services: Create and assign [IMS Profile](#)

- Configure roaming: Create and assign [Roaming Profile](#)
- Link physical SIM: Create and assign [SIM](#)

**See Also:**

- [Multi-MSISDN Documentation](#) - Assigning multiple phone numbers
  - [Profiles Documentation](#) - Advanced profile configuration
- 

## Complete Static IP Provisioning

This example demonstrates provisioning a subscriber with a static IP address from scratch.

**Scenario:** Provision an IoT device subscriber that needs a static IPv4 address on the "internet" APN.

```
# Prerequisites: jq must be installed (apt-get install jq or brew install jq)
```

```
# 1. Create Key Set
```

```
KEY_SET_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/key_set \  
-H "Content-Type: application/json" \  
-d '{  
  "ki": "0123456789ABCDEF0123456789ABCDEF",  
  "opc": "FEDCBA9876543210FEDCBA9876543210",  
  "authentication_algorithm": "milenage",  
  "amf": "8000",  
  "sqn": 0  
}' | jq -r '.response.id')
```

```
# 2. Create APN QoS Profile
```

```
APN_QOS_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/apn/qos_profile \  
-H "Content-Type: application/json" \  
-d '{  
  "name": "IoT Best Effort",  
  "allocation_retention_priority": 8,  
  "apn_ambr_dl_kbps": 10000,  
  "apn_ambr_ul_kbps": 5000,  
  "pre_emption_capability": false,  
  "pre_emption_vulnerability": false,  
  "qci": 9  
}' | jq -r '.response.id')
```

```
# 3. Create APN Identifier
```

```
APN_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/apn/identifier \  
-H "Content-Type: application/json" \  
-d '{  
  "apn": "internet",  
  "ip_version": "ipv4"  
}' | jq -r '.response.id')
```

```
# 4. Create APN Profile
```

```
APN_PROFILE_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/apn/profile \  
-H "Content-Type: application/json" \  
-d "{
```

```
\ "apn_identifier_id\ ": $APN_ID,  
\ "apn_qos_profile_id\ ": $APN_QOS_ID,  
\ "name\ ": \ "IoT Internet APN\  
}" | jq -r '.response.id')
```

# 5. Create Static IP for the APN

```
STATIC_IP_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/epc/static_ip \  
-H "Content-Type: application/json" \  
-d "{  
  \ "apn_profile_id\ ": $APN_PROFILE_ID,  
  \ "ipv4_static_ip\ ": \ "100.64.1.100\  
}" | jq -r '.response.id')
```

# 6. Create EPC Profile

```
EPC_PROFILE_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/epc/profile \  
-H "Content-Type: application/json" \  
-d "{  
  \ "apn_profiles\ ": [$APN_PROFILE_ID],  
  \ "name\ ": \ "IoT Data Plan\  
  \ "network_access_mode\ ": \ "packet_only\  
  \ "tracking_area_update_interval_seconds\ ": 600,  
  \ "ue_ambr_dl_kbps\ ": 10000,  
  \ "ue_ambr_ul_kbps\ ": 5000  
}" | jq -r '.response.id')
```

# 7. Create MSISDN (phone number)

```
MSISDN_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/msisdns \  
-H "Content-Type: application/json" \  
-d '{  
  "msisdns": "14155551000"  
}' | jq -r '.response.id')
```

# 8. Create Subscriber with Static IP

```
SUBSCRIBER_ID=$(curl -k -X POST  
https://hss.example.com:8443/api/subscriber \  
-H "Content-Type: application/json" \  
-d "{  
  \ "imsi\ ": \ "001001999999999\  
  \ "key_set_id\ ": $KEY_SET_ID,  
  \ "epc_profile_id\ ": $EPC_PROFILE_ID,  
  \ "msisdns\ ": [$MSISDN_ID],
```

```
\ "static_ips\": [${STATIC_IP_ID}
}" | jq -r '.response.id')
```

```
echo "IoT Subscriber provisioned successfully!"
echo " Subscriber ID: $SUBSCRIBER_ID"
echo " IMSI: 001001999999999"
echo " MSISDN: 14155551000"
echo " Static IPv4: 100.64.1.100 (on 'internet' APN)"
```

## What This Creates:

This provisioning workflow creates a complete IoT subscriber with:

1. **Cryptographic keys** ([Key Set](#)) - For authentication
2. **APN configuration** ([APN Profile](#)) - "internet" access point
3. **Static IP assignment** ([Static IP](#)) - Fixed IPv4 address 100.64.1.100
4. **Data service profile** ([EPC Profile](#)) - IoT-optimized bandwidth limits
5. **Phone number** ([MSISDN](#)) - For device identification
6. **Subscriber record** ([Subscriber](#)) - The complete subscriber entity

## Result:

When this subscriber attaches to the network and connects to the "internet" APN, they will receive the static IP address `100.64.1.100` instead of a dynamic DHCP address.

## Next Steps:

- Add additional APNs with static IPs: Repeat steps 2-5 for each APN
- Enable voice services: Create and assign [IMS Profile](#)
- Configure roaming: Create and assign [Roaming Profile](#)
- Link physical SIM: Create and assign [SIM](#)

## See Also:

- [Static IP Management](#) - Detailed static IP documentation
  - [Complete Subscriber Provisioning](#) - Basic provisioning without static IP
  - [Multi-MSISDN Documentation](#) - Assigning multiple phone numbers
-

[← Back to API Reference](#)

# OmniHSS API Reference

[← Back to Operations Guide](#)

---

## Table of Contents

- [API Overview](#)
  - [Key Set Management](#)
  - [Subscriber Management](#)
  - [Sh RepositoryData \(Transparent Data\)](#)
  - [MSISDN Management](#)
  - [SIM Management](#)
  - [Key Set Management](#)
  - [Profile Management](#)
  - [Static IP Management](#)
  - [Roaming Management](#)
  - [EIR Management](#)
  - [Status and Health](#)
  - [Error Handling](#)
  - [API Usage Examples](#)
- 

## API Overview

### Base URL

```
https://[hostname]:8443/api
```

# Request Format

- **Content-Type:** application/json
- **Protocol:** HTTPS only
- **Port:** 8443 (configurable)

**Important:** All API endpoints expect "flat" JSON payloads without wrapper objects.

## Correct Format:

```
{
  "name": "value",
  "field": "value"
}
```

## Incorrect Format (Do Not Use):

```
{
  "subscriber": {
    "name": "value",
    "field": "value"
  }
}
```

## Example:

```
# ✓ Correct
curl -X POST https://hss.example.com:8443/api/ims/profile \
  -H "Content-Type: application/json" \
  -d '{"name": "default", "ifc_template": "..."}'

# ✗ Incorrect
curl -X POST https://hss.example.com:8443/api/ims/profile \
  -H "Content-Type: application/json" \
  -d '{"ims_profile": {"name": "default", "ifc_template": "..."}}'
```

# Response Format

All responses are JSON with the following structure:

## Success Response:

```
{
  "status": "success",
  "response": { ... }
}
```

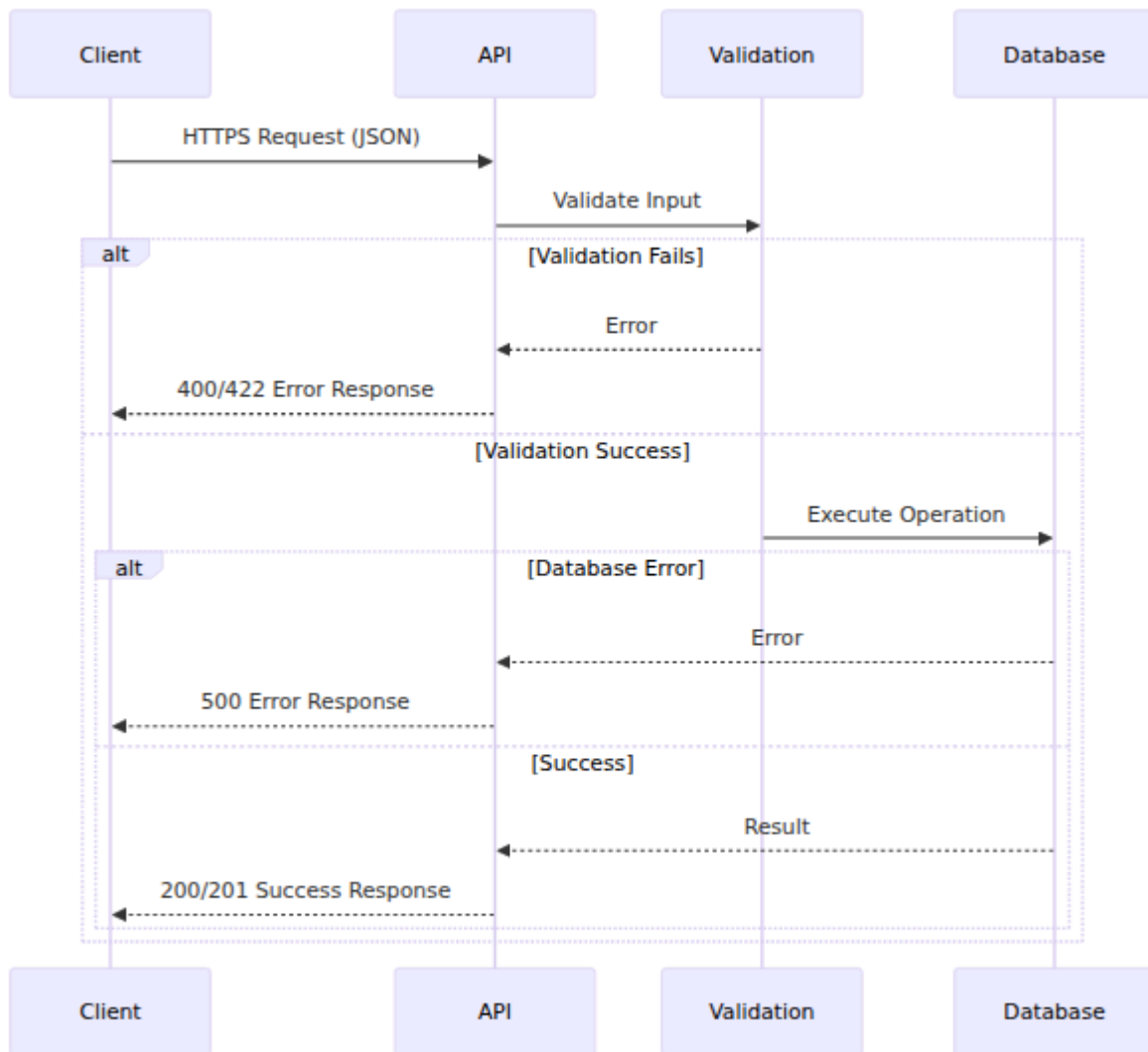
## Error Response:

```
{
  "status": "error",
  "response": {
    "invalid_fields": {
      "field_name": "error message"
    }
  }
}
```

# HTTP Status Codes

Code	Meaning	Use Case
200	OK	Successful GET, PUT, DELETE
201	Created	Successful POST
400	Bad Request	Invalid input data
404	Not Found	Resource doesn't exist
422	Unprocessable Entity	Validation error
500	Internal Server Error	Server-side error

# API Request Flow



---

# Subscriber Management

## List Subscribers

Retrieve all subscribers or filter by criteria.

**Endpoint:** `GET /api/subscriber`

**Query Parameters:**

Parameter	Type	Description
enabled	boolean	Filter by enabled status
ims_enabled	boolean	Filter by IMS enabled status

### Example Request:

```
curl -k https://hss.example.com:8443/api/subscriber
```

### Example Response:

```
{
  "data": [
    {
      "id": 1,
      "imsi": "001001123456789",
      "enabled": true,
      "ims_enabled": true,
      "sim_id": 1,
      "key_set_id": 1,
      "epc_profile_id": 1,
      "ims_profile_id": 1,
      "roaming_profile_id": 1,
      "custom_attributes": {},
      "inserted_at": "2025-10-15T10:30:00Z",
      "updated_at": "2025-10-15T10:30:00Z"
    }
  ]
}
```

## Get Subscriber by ID

Retrieve a specific subscriber by database ID.

**Endpoint:** GET /api/subscriber/:id

**Path Parameters:**

Parameter	Type	Description
<code>id</code>	integer	Subscriber database ID

**Example Request:**

```
curl -k https://hss.example.com:8443/api/subscriber/1
```

## Get Subscriber by IMSI

Retrieve a subscriber by their IMSI.

**Endpoint:** `GET /api/subscriber/imsi/:imsi`

**Path Parameters:**

Parameter	Type	Description	Format
<code>imsi</code>	string	International Mobile Subscriber Identity	14-15 digits

**Example Request:**

```
curl -k https://hss.example.com:8443/api/subscriber/imsi/001001123456789
```

**Use Case:** Troubleshooting a specific subscriber by their IMSI.

## Get Subscriber by MSISDN

Retrieve a subscriber by their phone number.

**Endpoint:** `GET /api/subscriber/msisdn/:msisdn`

**Path Parameters:**

Parameter	Type	Description	Format
msisdn	string	Mobile Station ISDN Number	1-15 digits (E.164)

### Example Request:

```
curl -k
https://hss.example.com:8443/api/subscriber/msisdn/14155551234
```

**Use Case:** Looking up subscriber information when you only have their phone number.

## Create Subscriber

Provision a new subscriber.

**Endpoint:** POST /api/subscriber

### Request Body:

```
{
  "subscriber": {
    "imsi": "001001123456789",
    "enabled": true,
    "ims_enabled": true,
    "sim_id": 1,
    "key_set_id": 1,
    "epc_profile_id": 1,
    "ims_profile_id": 1,
    "roaming_profile_id": 1,
    "custom_attributes": {
      "note": "Test subscriber"
    }
  }
}
```

### Required Fields:

- `imsi` - Must be 14-15 digits, unique
- `key_set_id` - Must reference existing [Key Set](#)
- `epc_profile_id` - Must reference existing [EPC Profile](#)

### Optional Fields:

- `enabled` - Default: true
- `ims_enabled` - Default: true
- `sim_id` - Reference to [SIM card](#)
- `ims_profile_id` - Reference to [IMS Profile](#) (required for IMS services)
- `roaming_profile_id` - Reference to [Roaming Profile](#) (required for roaming control)
- `msisdns` - Array of [MSISDN](#) IDs (phone numbers)
- `static_ips` - Array of [Static IP](#) IDs for APN assignments
- `custom_attributes` - Custom key-value pairs

### See Also:

- [Complete Subscriber Provisioning Example](#) - End-to-end workflow
- [Multi-MSISDN Documentation](#) - Assigning phone numbers to subscribers
- [Static IP Management](#) - Assigning static IPs to APNs

### Example Request:

```
curl -k -X POST https://hss.example.com:8443/api/subscriber \
-H "Content-Type: application/json" \
-d '{
  "subscriber": {
    "imsi": "001001123456789",
    "key_set_id": 1,
    "epc_profile_id": 1
  }
}'
```

### Provisioning Flow:

Start Provisioning

Key Set Exists?

No

Yes

Error: Key Set Not Found

EPC Profile Exists?

No

Yes

Error: EPC Profile Not Found

IMSI Unique?

No

Yes

Error: IMSI Already Exists

Create Subscriber

Auto-Crete Subscriber State

201 Created

# Update Subscriber

Modify an existing subscriber.

**Endpoint:** PUT /api/subscriber/:id

## Path Parameters:

Parameter	Type	Description
id	integer	Subscriber database ID

## Request Body:

```
{
  "subscriber": {
    "enabled": false,
    "ims_enabled": false,
    "epc_profile_id": 2,
    "custom_attributes": {
      "note": "Temporarily disabled"
    }
  }
}
```

## Updatable Fields:

- enabled - Enable/disable all services
- ims\_enabled - Enable/disable IMS services
- sim\_id - Change SIM card assignment
- key\_set\_id - Change crypto keys (be careful!)
- epc\_profile\_id - Change data service profile
- ims\_profile\_id - Change voice service profile
- roaming\_profile\_id - Change roaming policy
- msisdns - Update phone numbers assigned to subscriber
- static\_ips - Update static IP assignments to APNs
- custom\_attributes - Update custom data

## Not Updatable:

- `imsi` - Cannot change IMSI (delete and recreate instead)

## See Also:

- [Profile Management](#) - Managing service profiles

## Example Request:

```
curl -k -X PUT https://hss.example.com:8443/api/subscriber/1 \
-H "Content-Type: application/json" \
-d '{
  "subscriber": {
    "enabled": false
  }
}'
```

## Use Cases:

- Temporarily disable subscriber: `{"enabled": false}`
- Disable voice services only: `{"ims_enabled": false}`
- Change service profile: `{"epc_profile_id": 2}` (see [EPC Profiles](#))
- Update roaming policy: `{"roaming_profile_id": 3}` (see [Roaming Management](#))

## Delete Subscriber

Remove a subscriber from the system.

**Endpoint:** `DELETE /api/subscriber/:id`

### Path Parameters:

Parameter	Type	Description
<code>id</code>	integer	Subscriber database ID

### Example Request:

```
curl -k -X DELETE https://hss.example.com:8443/api/subscriber/1
```

**Warning:** This permanently deletes the subscriber and all associated state data (PDN sessions, calls, etc.). The IMSI can be reused after deletion.

**Note:** Deleting a subscriber does NOT delete the associated:

- **Key Set** - Can be reused for other subscribers
- **SIM** - Can be reassigned to a new subscriber
- **Profiles** - Shared resources used by multiple subscribers
- **MSISDNs** - Must be deleted separately if desired

## Cancel Location Request (Force Detach)

Send a Cancel Location Request (CLR) to force detach a subscriber from their currently registered MME.

**Endpoint:** `POST /api/subscriber/cancel_location`

### Request Body:

```
{
  "imsi": "001001123456789"
}
```

### Parameters:

Parameter	Type	Required	Description
<code>imsi</code>	string	Yes	IMSI of subscriber to detach (14-15 digits)

### Example Request:

```
curl -k -X POST
https://hss.example.com:8443/api/subscriber/cancel_location \
-H "Content-Type: application/json" \
-d '{"imsi": "001001123456789"}'
```

### Success Response (200 OK):

```
{
  "data": {
    "message": "Cancel Location Request sent successfully",
    "imsi": "001001123456789",
    "destination_host": "mme01.operator.com",
    "destination_realm": "epc.operator.com"
  }
}
```

### Error Response (404 Not Found):

```
{
  "error": "Subscriber not found or not currently registered at
any MME"
}
```

### Behavior:

- Sends S6a CLR to the MME where subscriber is currently registered (`subscriber_state.last_seen_mme`)
- Uses `Cancellation-Type: subscription_withdrawal` (forces full detach)
- Sets `CLR-Flags: {s6a_indicator: 1, reattach_required: 1}` (UE must re-authenticate)
- Returns 404 if subscriber has never registered or `last_seen_mme` is null
- **Affects all MSISDNs** associated with the IMSI (same physical device/SIM)

### Use Cases:

- **Fraud Prevention:** Immediately detach suspicious subscriber
- **Subscription Termination:** Force logout when account is disabled

- **Troubleshooting:** Clear stale MME registration for debugging
- **Migration:** Force re-authentication to apply new profile settings
- **Security:** Immediately disconnect compromised subscriber

## Multi-IMSI Considerations:

When using CLR with multi-MSISDN scenarios:

### 1. Multiple MSISDNs, Single IMSI:

```
// Subscriber has IMSI 001001123456789 with MSISDNs  
["+1234567890", "+9876543210"]  
POST /api/subscriber/cancel_location  
{  
  "imsi": "001001123456789"  
}  
  
// Result: One CLR sent, both MSISDNs affected (same device)
```

### 2. Different IMSIs (Different Devices):

```
// Two subscribers with same MSISDN but different IMSIs (number  
porting scenario)  
// Subscriber A: IMSI 0010011111111111, MSISDN "+1234567890"  
// Subscriber B: IMSI 0010012222222222, MSISDN "+1234567890"  
  
POST /api/subscriber/cancel_location  
{  
  "imsi": "0010011111111111"  
}  
  
// Result: Only Subscriber A detached, Subscriber B unaffected
```

## Important Notes:

- **IMSI-based:** CLR is always sent per IMSI, not per MSISDN
- **Asynchronous:** CLR is sent asynchronously; success response means CLR was sent, not that MME processed it
- **No validation of MME status:** CLR is sent even if MME is unreachable (standard HSS behavior)
- **Idempotent:** Safe to call multiple times for same IMSI

## Related Documentation:

- [Cancel Location Request Protocol Flow](#)
  - [Multi-IMSI Scenarios](#)
  - [S6a Interface Architecture](#)
- 

# Sh RepositoryData (Transparent Data)

Provisioning access to a subscriber's Sh transparent documents (MMTEL-Services, IMS-ODB-Information, ...) — the same opaque documents an Application Server reads and writes over the Diameter Sh interface — via `GET/PUT/DELETE /api/subscriber/repository_data/:imsi`.

See [Sh RepositoryData \(Transparent Data\)](#) for the data model, message flows, SequenceNumber concurrency rules, and the full REST API reference.

---

## MSISDN Management

MSISDNs (phone numbers) can be assigned to subscribers to enable voice services. See [Multi-MSISDN Documentation](#) for details on assigning multiple numbers to a single subscriber.

### List MSISDNs

Retrieve all phone numbers.

**Endpoint:** `GET /api/msisdn`

**Example Request:**

```
curl -k https://hss.example.com:8443/api/msisdn
```

## Get MSISDN

Retrieve a specific phone number.

**Endpoint:** GET /api/msisdn/:id

**Example Request:**

```
curl -k https://hss.example.com:8443/api/msisdn/1
```

## Create MSISDN

Create a new phone number.

**Endpoint:** POST /api/msisdn

**Request Body:**

```
{
  "msisdn": {
    "msisdn": "14155551234"
  }
}
```

**Validation:**

- Must be 1-15 digits
- Must be unique
- Must follow E.164 format (international format without + sign)

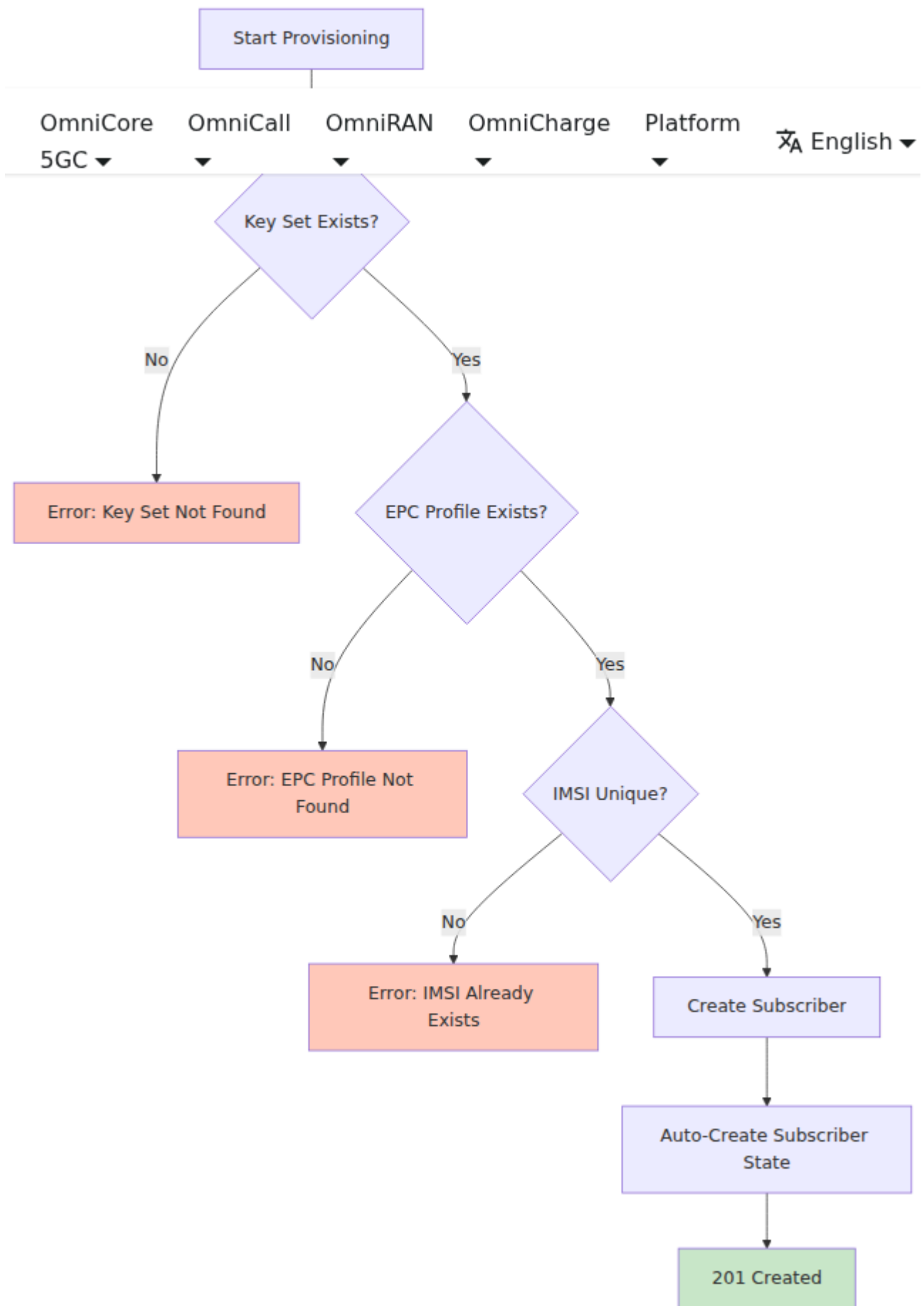
**Example Request:**

```
curl -k -X POST https://hss.example.com:8443/api/msisdn \  
-H "Content-Type: application/json" \  
-d '{  
  "msisdn": {  
    "msisdn": "14155551234"  
  }  
'
```

## Assign MSISDN to Subscriber

To assign a phone number to a subscriber, you need to create a join record. This is typically done through the subscriber update endpoint or via direct database manipulation.

### Multi-MSISDN Pattern:



See [Multi-MSISDN and Multi-IMSI Features](#) for detailed usage.

## Delete MSISDN

Remove a phone number.

**Endpoint:** DELETE /api/msisdn/:id

**Example Request:**

```
curl -k -X DELETE https://hss.example.com:8443/api/msisdn/1
```

---

## SIM Management

SIM card records store physical SIM card information including ICCID, vendor details, PIN/PUK codes, and OTA keys. SIM records can optionally be linked to [subscribers](#).

**See Also:**

- [Multi-IMSI Documentation](#) - Multiple subscribers on one physical SIM

## List SIMs

Retrieve all SIM cards.

**Endpoint:** GET /api/sim

**Example Request:**

```
curl -k https://hss.example.com:8443/api/sim
```

## Get SIM

Retrieve a specific SIM card.

**Endpoint:** GET /api/sim/:id

## Example Request:

```
curl -k https://hss.example.com:8443/api/sim/1
```

## Create SIM

Create a new SIM card record.

**Endpoint:** `POST /api/sim`

### Request Body:

```
{
  "sim": {
    "iccid": "8991101200003204510",
    "sim_vendor": "Gemalto",
    "batch_name": "2025-Q1-Batch-01",
    "is_esim": false,
    "pin1": "1234",
    "pin2": "5678",
    "puk1": "12345678",
    "puk2": "87654321",
    "adm1": "admin-code-1",
    "kic": "0123456789ABCDEF0123456789ABCDEF",
    "kid": "FEDCBA9876543210FEDCBA9876543210"
  }
}
```

### Required Fields:

- `iccid` - 19-20 digits, unique

### Optional but Important Fields:

- `sim_vendor` - Manufacturer name
- `batch_name` - For tracking
- `is_esim` - Boolean flag for eSIM
- `pin1`, `pin2` - End-user PIN codes

- `puk1`, `puk2` - PIN unlock codes
- `adm1-adm10` - Administrative codes
- `kic`, `kid` - OTA security keys (hex string)

### Example Request:

```
curl -k -X POST https://hss.example.com:8443/api/sim \  
-H "Content-Type: application/json" \  
-d '{  
  "sim": {  
    "iccid": "8991101200003204510",  
    "sim_vendor": "Gemalto"  
  }  
'
```

## Update SIM

Modify SIM card data.

**Endpoint:** `PUT /api/sim/:id`

### Example Request:

```
curl -k -X PUT https://hss.example.com:8443/api/sim/1 \  
-H "Content-Type: application/json" \  
-d '{  
  "sim": {  
    "batch_name": "Updated-Batch-Name"  
  }  
'
```

## Delete SIM

Remove a SIM card record.

**Endpoint:** `DELETE /api/sim/:id`

**Warning:** Ensure no subscribers reference this SIM before deleting.

---

# Key Set Management

Key sets contain the cryptographic material (Ki, OPC/OP, AMF, SQN) used for subscriber authentication via the Milenage algorithm. Each **subscriber** must reference a key set.

## See Also:

- **Protocol Flows** - Authentication procedures using key sets

## List Key Sets

Retrieve all cryptographic key sets.

**Endpoint:** GET /api/key\_set

### Example Request:

```
curl -k https://hss.example.com:8443/api/key_set
```

## Get Key Set

Retrieve a specific key set.

**Endpoint:** GET /api/key\_set/:id

### Example Request:

```
curl -k https://hss.example.com:8443/api/key_set/1
```

### Response Example:

```
{
  "data": {
    "id": 1,
    "ki": "0123456789ABCDEF0123456789ABCDEF",
    "opc": "FEDCBA9876543210FEDCBA9876543210",
    "op": null,
    "amf": "8000",
    "sqn": 0,
    "authentication_algorithm": "milenage",
    "ota_counter": 0
  }
}
```

## Create Key Set

Create a new cryptographic key set.

**Endpoint:** `POST /api/key_set`

### Request Body:

```
{
  "key_set": {
    "ki": "0123456789ABCDEF0123456789ABCDEF",
    "opc": "FEDCBA9876543210FEDCBA9876543210",
    "amf": "8000",
    "sqn": 0,
    "authentication_algorithm": "milenage"
  }
}
```

### Required Fields:

- `ki` - 128-bit key (32 hex characters)
- Either `opc` OR `op` (OPC can be derived from OP)
- `authentication_algorithm` - Currently only "milenage"

### Optional Fields:

- `amf` - Default: "8000"
- `sqn` - Default: 0
- `ota_counter` - Default: 0

### Key Format:

- All keys are hexadecimal strings
- Ki, OPC, OP: 32 hex characters (128 bits)
- AMF: 4 hex characters (16 bits)

### Example Request:

```
curl -k -X POST https://hss.example.com:8443/api/key_set \
-H "Content-Type: application/json" \
-d '{
  "key_set": {
    "ki": "0123456789ABCDEF0123456789ABCDEF",
    "opc": "FEDCBA9876543210FEDCBA9876543210",
    "authentication_algorithm": "milena"
  }
}'
```

**Security Warning:** Key sets contain highly sensitive cryptographic material. Protect API access accordingly.

## Update Key Set

Modify an existing key set.

**Endpoint:** `PUT /api/key_set/:id`

**Warning:** Changing keys for an active **subscriber** will cause authentication failures. Only update keys during maintenance windows or for new subscribers.

**Impact:** Updates affect all subscribers using this key set immediately. Active subscribers will fail authentication on next attach attempt.

# Delete Key Set

Remove a key set.

**Endpoint:** DELETE /api/key\_set/:id

**Warning:** Ensure no [subscribers](#) reference this key set before deleting. Query subscribers first to check for references.

---

# Profile Management

## EPC Profiles

EPC (Evolved Packet Core) profiles define data service parameters for subscribers. These profiles are referenced when creating [subscribers](#).

### List EPC Profiles

**Endpoint:** GET /api/epc/profile

### Get EPC Profile

**Endpoint:** GET /api/epc/profile/:id

### Create EPC Profile

**Endpoint:** POST /api/epc/profile

**Request Body:**

```

{
  "apn_profiles": [],
  "name": "Standard Data Plan",
  "network_access_mode": "packet_only",
  "tracking_area_update_interval_seconds": 600,
  "ue_ambr_dl_kbps": 100000,
  "ue_ambr_ul_kbps": 50000
}

```

### Fields:

Field	Description	Units	Type
name	Profile name	Text	Unique
ue_ambr_dl_kbps	Download bandwidth limit	Kbps	1000
ue_ambr_ul_kbps	Upload bandwidth limit	Kbps	5000
network_access_mode	Access type	String	"packet_only" or "packet_offload"
tracking_area_update_interval_seconds	TAU timer	Seconds	600 (min)
apn_profiles	List of APN profile IDs	Array	[] or [1, 2, 3]

### Example Request:

```
curl -k -X POST https://hss.example.com:8443/api/epc/profile \
-H "Content-Type: application/json" \
-d '{
  "apn_profiles": [],
  "name": "Premium 100Mbps",
  "network_access_mode": "packet_only",
  "tracking_area_update_interval_seconds": 600,
  "ue_ambr_dl_kbps": 100000,
  "ue_ambr_ul_kbps": 50000
}'
```

### See Also:

- [Profiles Documentation](#) - Detailed profile configuration guide
- [Complete Subscriber Provisioning](#) - Using EPC profiles in provisioning

### Update EPC Profile

**Endpoint:** `PUT /api/epc/profile/:id`

**Note:** Changes to EPC profiles affect all [subscribers](#) using this profile. Active sessions may need to be re-established.

### Delete EPC Profile

**Endpoint:** `DELETE /api/epc/profile/:id`

**Warning:** Ensure no [subscribers](#) reference this profile before deleting.

## IMS Profiles

IMS (IP Multimedia Subsystem) profiles define voice service parameters and Initial Filter Criteria (IFC) for subscribers. These profiles are referenced when creating [subscribers](#) with IMS services enabled.

### List IMS Profiles

**Endpoint:** `GET /api/ims/profile`

### Create IMS Profile

**Endpoint:** POST /api/ims/profile

### Request Body:

```
{
  "name": "Standard VoLTE",
  "ifc_template": "<IMS-XML-Template-Here>"
}
```

### Required Fields:

- `name` - Profile name (must be unique)
- `ifc_template` - IFC (Initial Filter Criteria) XML template with Liquid template variables

### IFC Template Variables:

The IFC template supports the following Liquid template variables that are dynamically substituted:

Variable	Description	Example Value
<code>{{ imsi }}</code>	Subscriber IMSI	001001123456789
<code>{{ msisdns }}</code>	Array of MSISDNs (for loops)	<code>["14155551234", "14155555678"]</code>
<code>{{ mcc }}</code>	Mobile Country Code	001
<code>{{ mnc }}</code>	Mobile Network Code	001

### How Template Rendering Works:

The IFC template is stored as a **Liquid template** (similar to Jinja2) and is rendered **dynamically** during IMS operations:

1. **Storage:** When you create an IMS profile, the template is stored as-is with variables like `{{ imsi }}` and `{% for msisdn in msisdns %}`

2. **Validation:** The API validates the template by rendering it with test data to ensure valid XML syntax
3. **Runtime Rendering:** When a subscriber performs IMS registration (MAA/SAA), the HSS:
  - Retrieves the subscriber's IMS profile
  - Renders the template with the subscriber's actual data:
    - `{{ imsi }}` → subscriber's IMSI
    - `{{ msisdns }}` → subscriber's phone numbers
    - `{{ mcc }}` → configured Mobile Country Code
    - `{{ mnc }}` → configured Mobile Network Code
  - Returns the rendered XML to the S-CSCF via Cx/Diameter

### Template Syntax:

```
<!-- Simple variable substitution -->
{{ imsi }}

<!-- For loops over arrays -->
{% for msisdn in msisdns %}
  <MSISDN>{{ msisdn }}</MSISDN>
{% endfor %}

<!-- Combining variables -->
{{ imsi }}@ims.mnc{{ mnc }}.mcc{{ mcc }}.3gppnetwork.org
```

### IFC Template Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<IMSSubscription>
<PrivateID>{{ imsi }}@ims.mnc{{ mnc }}.mcc{{ mcc
}}.3gppnetwork.org</PrivateID>
<ServiceProfile>
{% for msisdn in msisdns %}
<PublicIdentity>
<Identity>sip:{{ msisdn }}@ims.mnc{{ mnc }}.mcc{{ mcc
}}.3gppnetwork.org</Identity>
<Extension>
<IdentityType>0</IdentityType>
</Extension>
</PublicIdentity>
<PublicIdentity>
<Identity>tel:{{ msisdn }}</Identity>
<Extension>
<IdentityType>0</IdentityType>
</Extension>
</PublicIdentity>
{% endfor %}
<InitialFilterCriteria>
<Priority>10</Priority>
<TriggerPoint>
<ConditionTypeCNF>0</ConditionTypeCNF>
<SPT>
<ConditionNegated>0</ConditionNegated>
<Group>0</Group>
<Method>REGISTER</Method>
</SPT>
</TriggerPoint>
<ApplicationServer>
<ServerName>sip:as.ims.mnc{{ mnc }}.mcc{{ mcc
}}.3gppnetwork.org</ServerName>
<DefaultHandling>0</DefaultHandling>
</ApplicationServer>
</InitialFilterCriteria>
</ServiceProfile>
</IMSSubscription>

```

### Example Request (curl):

```
curl -k -X POST https://hss.example.com:8443/api/ims/profile \
  -H "Content-Type: application/json" \
  -d '{
    "name": "default",
    "ifc_template": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<IMSSubscription><ServiceProfile>...</ServiceProfile>
</IMSSubscription>"
  }'
```

### Example Request (Python):

```
import requests

response = requests.post(
    "https://hss.example.com:8443/api/ims/profile",
    json={
        "name": "default",
        "ifc_template": ifc_template_string
    },
    verify=False # For self-signed certificates
)
```

### Success Response (201 Created):

```
{
  "status": "success",
  "response": {
    "id": 1,
    "name": "default",
    "ifc_template": "<?xml version=\"1.0\" encoding=\"UTF-8\"?
>...\"
  }
}
```

### Validation:

- The API validates that the IFC template is valid XML
- Template variables are rendered with test data to verify syntax
- The `name` field must be unique and non-empty

## See Also:

- [Profiles Documentation](#) - IFC template details and examples
- [Protocol Flows](#) - IMS registration and call flows
- [Default IFC Template](#) - Reference implementation

## APN Profiles

APN (Access Point Name) profiles consist of three components that work together:

1. **APN Identifier** - Defines the APN name and IP version
2. **APN QoS Profile** - Defines Quality of Service parameters
3. **APN Profile** - Combines identifier and QoS, linked to [EPC Profiles](#)

See [PCRF Documentation](#) for detailed policy configuration, QoS management, and automatic re-auth. See also [Profiles Documentation](#) for APN configuration examples.

### List APN Identifiers

**Endpoint:** `GET /api/apn/identifier`

### Create APN Identifier

**Endpoint:** `POST /api/apn/identifier`

### Request Body:

```
{
  "apn": "internet",
  "ip_version": "ipv4v6"
}
```

### IP Version Values:

- `"ipv4"` - IPv4 only
- `"ipv6"` - IPv6 only

- "ipv4v6" - IPv4v6 (dual stack)
- "ipv4\_or\_ipv6" - IPv4 or IPv6 (network choice)

## List APN QoS Profiles

**Endpoint:** GET /api/apn/qos\_profile

## Create APN QoS Profile

**Endpoint:** POST /api/apn/qos\_profile

### Request Body:

```
{
  "name": "Best Effort Internet",
  "allocation_retention_priority": 8,
  "apn_ambr_dl_kbps": 50000,
  "apn_ambr_ul_kbps": 25000,
  "pre_emption_capability": false,
  "pre_emption_vulnerability": true,
  "qci": 9
}
```

## List APN Profiles

**Endpoint:** GET /api/apn/profile

## Create APN Profile

**Endpoint:** POST /api/apn/profile

### Request Body:

```
{
  "apn_identifier_id": 1,
  "apn_qos_profile_id": 1,
  "name": "Internet APN"
}
```

### Required Fields:

- `apn_identifier_id` - Must reference existing [APN Identifier](#)
- `apn_qos_profile_id` - Must reference existing [APN QoS Profile](#)

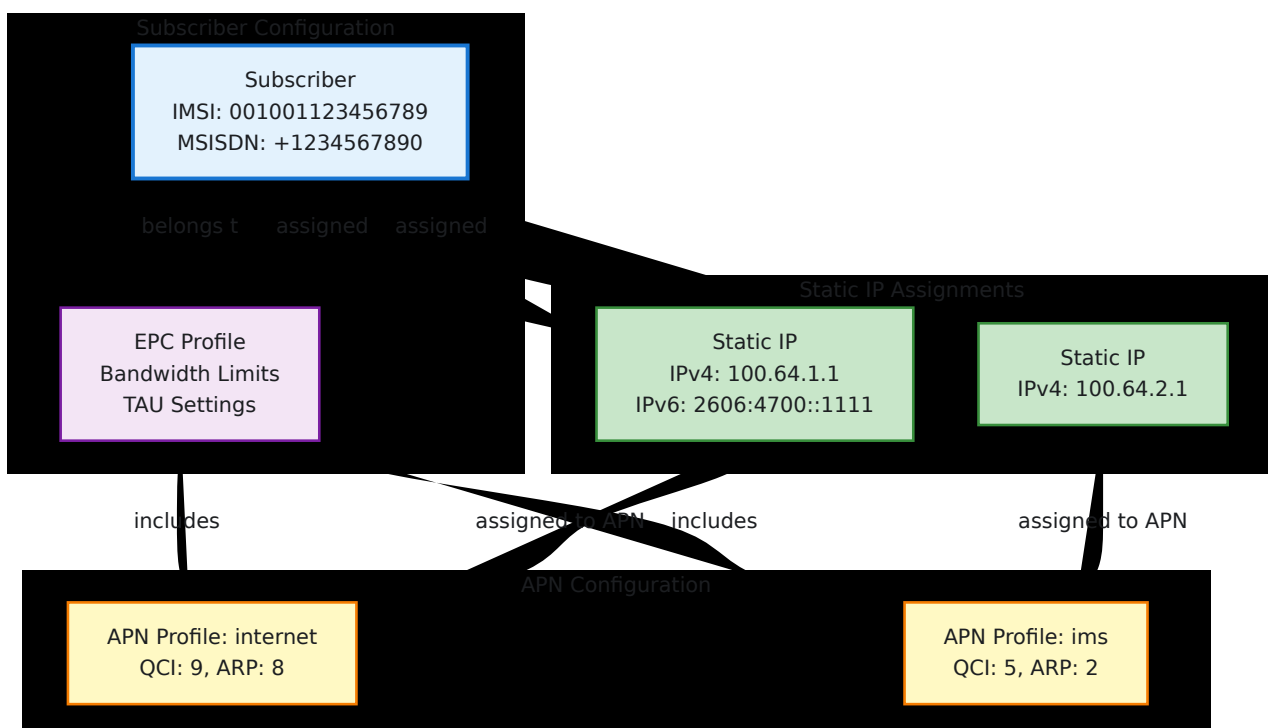
**See Also:**

- [Complete Subscriber Provisioning](#) - Full example including APN setup
- [EPC Profiles](#) - APN profiles are linked to EPC profiles

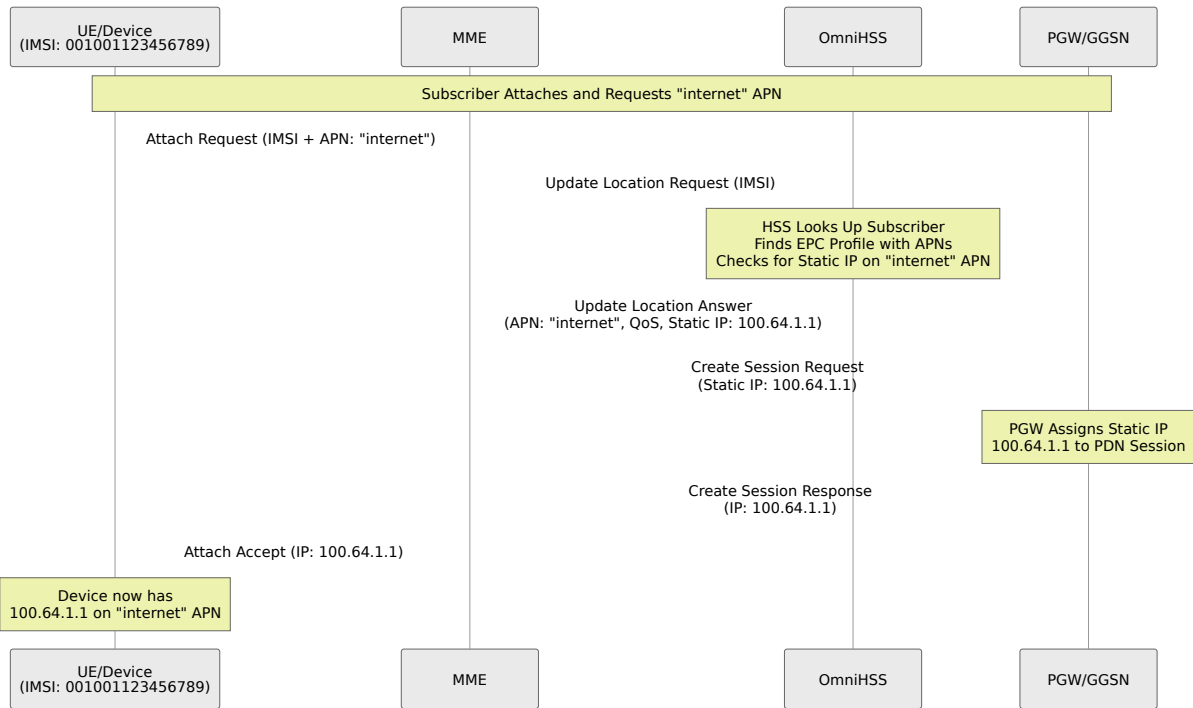
# Static IP Management

Static IP addresses can be assigned to specific APNs for individual subscribers. This allows subscribers to receive a predetermined IPv4 and/or IPv6 address when connecting to a particular APN, rather than receiving a dynamic address from a DHCP pool.

**Architecture:**

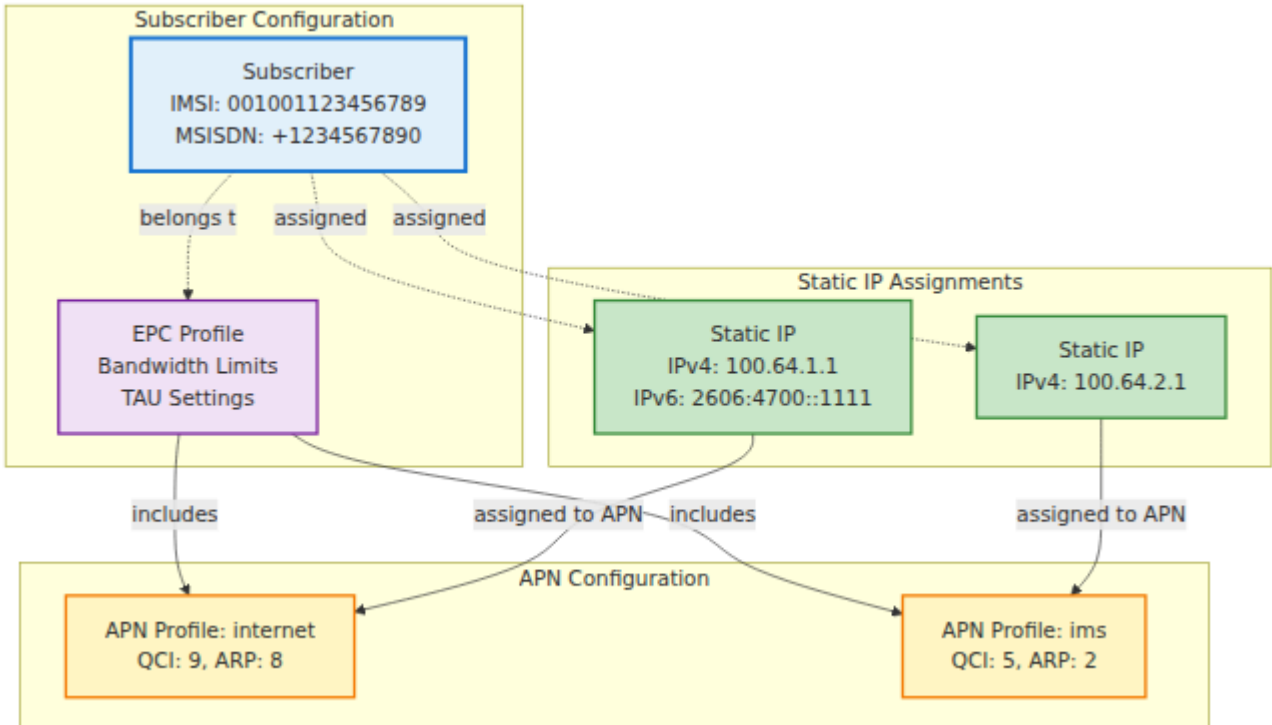


**Data Flow When Subscriber Connects:**



### Update Location Answer - APN Configuration Data Mapping:

This diagram shows exactly where each field in the S6a Update Location Answer APN-Configuration AVP comes from in the database:

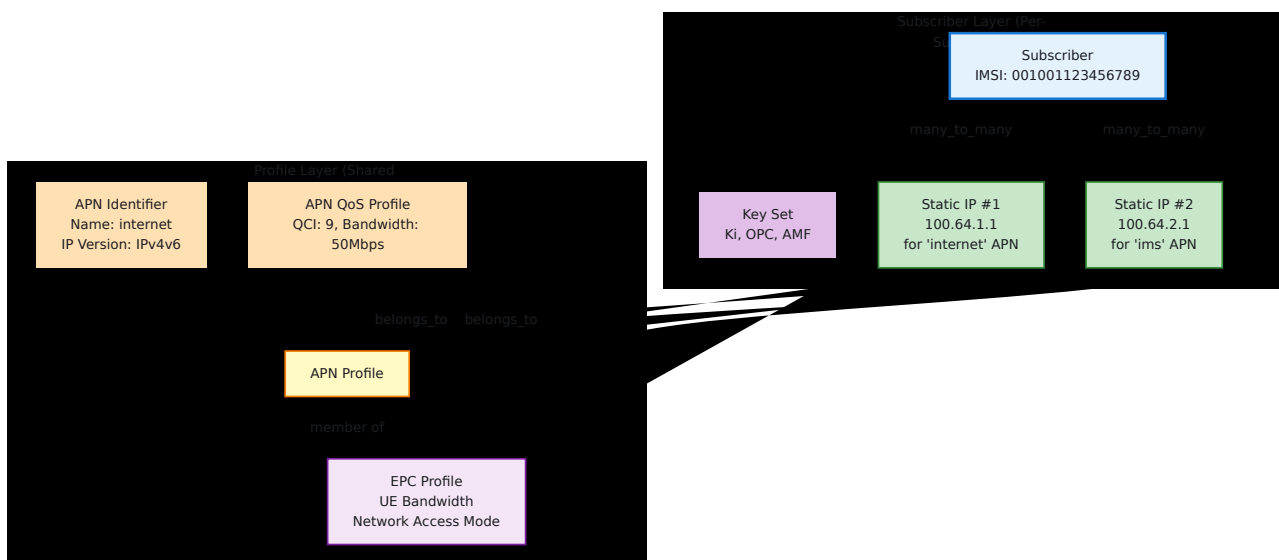


### Key Observations:

1. **Context-Identifier:** Sequential index (0, 1, 2...) for each APN in the profile

2. **Service-Selection:** Comes directly from `apn_identifier.apn` (e.g., "internet", "ims")
3. **PDN-Type:** Encoded from `apn_identifier.ip_version` (ipv4=0, ipv6=1, ipv4v6=2, ipv4\_or\_ipv6=3)
4. **QoS Parameters:** All from `apn_qos_profile` table
5. **AMBR Bandwidth:** Values are multiplied by 1000 (kbps → bps conversion)
6. **Served-Party-IP-Address:** Only included if static IP exists for this subscriber+APN combination
  - Lookup process: `subscriber.static_ips` → filter by `apn_profile_id` → extract IPs
  - IP version compatibility checked against `apn_identifier.ip_version`
7. **VPLMN-Dynamic-Address-Allowed:** Hardcoded to 0 (not allowed) - forces use of static IP if provided

### Relationship Hierarchy:



### Key Concepts:

- **Per-APN Assignment:** Each Static IP is linked to a specific **APN Profile**
- **One IP per APN per Subscriber:** A subscriber can only have one static IP assignment per APN
- **IPv4 and IPv6 Support:** Static IPs can be IPv4-only, IPv6-only, or dual-stack
- **Global IP Uniqueness:** Each IP address must be globally unique across **all** static IP records in the system

- The same IPv4 or IPv6 address cannot be assigned to multiple subscribers (even on different APNs)
- This prevents routing conflicts and IP address ambiguity
- Enforced by database unique indexes on `ipv4_static_ip` and `ipv6_static_ip` fields
- **Many-to-Many Relationship:** Subscribers and Static IPs are linked via a join table

### Use Cases:

- Fixed IP addresses for IoT devices
- Server hosting on mobile devices (requires static IP for inbound connections)
- Legacy applications that require specific IP addresses
- Network policy routing based on source IP
- Regulatory compliance requiring IP address tracking

## List Static IPs

Retrieve all static IP assignments.

**Endpoint:** `GET /api/epc/static_ip`

### Example Request:

```
curl -k https://hss.example.com:8443/api/epc/static_ip
```

### Example Response:

```
{
  "data": [
    {
      "id": 1,
      "apn_profile_id": 5,
      "ipv4_static_ip": "100.64.1.1",
      "ipv6_static_ip": "2606:4700:4700::1111",
      "apn_profile": {
        "id": 5,
        "name": "Internet APN",
        "apn_identifier": {
          "apn": "internet",
          "ip_version": "ipv4v6"
        }
      },
      "inserted_at": "2025-11-15T10:30:00Z",
      "updated_at": "2025-11-15T10:30:00Z"
    }
  ]
}
```

## Get Static IP

Retrieve a specific static IP assignment.

**Endpoint:** GET /api/epc/static\_ip/:id

### Path Parameters:

Parameter	Type	Description
id	integer	Static IP database ID

### Example Request:

```
curl -k https://hss.example.com:8443/api/epc/static_ip/1
```

# Create Static IP

Create a new static IP assignment for an APN.

**Endpoint:** `POST /api/epc/static_ip`

## Request Body:

```
{
  "static_ip": {
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1",
    "ipv6_static_ip": "2606:4700:4700::1111"
  }
}
```

## Required Fields:

- `apn_profile_id` - Must reference an existing [APN Profile](#)
- At least one of `ipv4_static_ip` OR `ipv6_static_ip` must be specified

## Optional Fields:

- `ipv4_static_ip` - IPv4 address (dotted decimal notation)
- `ipv6_static_ip` - IPv6 address (standard notation)

## IP Format Validation:

- IPv4: Standard dotted decimal format (e.g., `100.64.1.1`)
- IPv6: Standard colon-separated hexadecimal format (e.g., `2606:4700:4700::1111`)
- Both IPv4 and IPv6 addresses must be **globally unique across all static IP records**
  - This prevents IP address conflicts in the network
  - The same IP cannot be assigned to multiple subscribers, even on different APNs
  - This is a database-level constraint enforced by unique indexes

## Configuration Options:

Configuration	IPv4	IPv6	Example
<b>IPv4 Only</b>	✓	-	<code>{"ipv4_static_ip": "100.64.1.1"}</code>
<b>IPv6 Only</b>	-	✓	<code>{"ipv6_static_ip": "2606:4700:4700::1111"}</code>
<b>Dual Stack</b>	✓	✓	Both fields specified

### Example Requests:

#### IPv4-only Static IP:

```
curl -k -X POST https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1"
  }
}'
```

#### IPv6-only Static IP:

```
curl -k -X POST https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 6,
    "ipv6_static_ip": "2606:4700:4700::1111"
  }
}'
```

#### Dual-stack Static IP:

```
curl -k -X POST https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1",
    "ipv6_static_ip": "2606:4700:4700::1111"
  }
}'
```

### Success Response (201 Created):

```
{
  "data": {
    "id": 1,
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1",
    "ipv6_static_ip": "2606:4700:4700::1111",
    "inserted_at": "2025-11-15T10:30:00Z",
    "updated_at": "2025-11-15T10:30:00Z"
  }
}
```

### See Also:

- [Assign Static IP to Subscriber](#) - How to link this to a subscriber
- [APN Profiles](#) - Managing APN configurations

## Update Static IP

Modify an existing static IP assignment.

**Endpoint:** `PUT /api/epc/static_ip/:id`

**Path Parameters:**

Parameter	Type	Description
id	integer	Static IP database ID

### Request Body:

```
{
  "static_ip": {
    "ipv4_static_ip": "100.64.1.2",
    "ipv6_static_ip": "2606:4700:4700::1112"
  }
}
```

### Updatable Fields:

- `ipv4_static_ip` - Change IPv4 address
- `ipv6_static_ip` - Change IPv6 address
- `apn_profile_id` - Change APN assignment

### Not Updatable:

- `id` - Primary key (read-only)

**Warning:** Changing the IP address for an active subscriber will affect their next PDN connection. Active PDN sessions will continue to use the old IP until they disconnect and reconnect.

### Example Request:

```
curl -k -X PUT https://hss.example.com:8443/api/epc/static_ip/1 \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "ipv4_static_ip": "100.64.1.2"
  }
}'
```

# Delete Static IP

Remove a static IP assignment.

**Endpoint:** DELETE /api/epc/static\_ip/:id

## Path Parameters:

Parameter	Type	Description
id	integer	Static IP database ID

## Example Request:

```
curl -k -X DELETE https://hss.example.com:8443/api/epc/static_ip/1
```

## Behavior:

- Removes the static IP assignment
- Does NOT affect the [APN Profile](#) (APN remains available for other subscribers)
- Subscribers using this static IP will receive dynamic IPs on next connection
- The IP address becomes available for reuse after deletion

**Warning:** If a subscriber is actively using this static IP, deleting it will cause them to receive a dynamic IP on their next PDN connection. Ensure subscribers are offline or send a [Cancel Location Request](#) before deleting.

# Assign Static IP to Subscriber

To assign a static IP to a subscriber, you need to associate the Static IP record with the [Subscriber](#) during creation or update.

## Assignment Pattern:

1. **Create the Static IP** (see [Create Static IP](#))
2. **Assign to Subscriber** using the `static_ips` field

## Create Subscriber with Static IP:

```
# Step 1: Create static IP for "internet" APN
STATIC_IP_ID=$(curl -k -X POST
https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1",
    "ipv6_static_ip": "2606:4700:4700::1111"
  }
}' | jq -r '.data.id')

# Step 2: Create subscriber with static IP assigned
curl -k -X POST https://hss.example.com:8443/api/subscriber \
-H "Content-Type: application/json" \
-d "{
  \"subscriber\": {
    \"imsi\": \"001001123456789\",
    \"key_set_id\": 1,
    \"epc_profile_id\": 1,
    \"static_ips\": [\$STATIC_IP_ID]
  }
}"
```

## Update Existing Subscriber with Static IP:

```
curl -k -X PUT https://hss.example.com:8443/api/subscriber/1 \
-H "Content-Type: application/json" \
-d '{
  "subscriber": {
    "static_ips": [1, 2]
  }
}'
```

## Multiple Static IPs (Different APNs):

A subscriber can have multiple static IPs as long as each is for a different APN:

```

# Create static IP for "internet" APN
INTERNET_IP=$(curl -k -X POST
https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 5,
    "ipv4_static_ip": "100.64.1.1"
  }
}' | jq -r '.data.id')

# Create static IP for "ims" APN
IMS_IP=$(curl -k -X POST
https://hss.example.com:8443/api/epc/static_ip \
-H "Content-Type: application/json" \
-d '{
  "static_ip": {
    "apn_profile_id": 6,
    "ipv4_static_ip": "100.64.2.1"
  }
}' | jq -r '.data.id')

# Assign both to subscriber
curl -k -X POST https://hss.example.com:8443/api/subscriber \
-H "Content-Type: application/json" \
-d "{
  \"subscriber\": {
    \"imsi\": \"001001123456789\",
    \"key_set_id\": 1,
    \"epc_profile_id\": 1,
    \"static_ips\": [${INTERNET_IP}, ${IMS_IP}]
  }
}"

```

### Validation Rules:

- ✓ **Allowed:** Multiple static IPs for different APNs
- ✗ **Rejected:** Multiple static IPs for the same APN

### Error Example - Duplicate APN:

```
# This will FAIL if both static IPs reference the same APN
curl -k -X POST https://hss.example.com:8443/api/subscriber \
-H "Content-Type: application/json" \
-d '{
  "subscriber": {
    "imsi": "001001123456789",
    "static_ips": [1, 2]
  }
}'

# Error Response:
{
  "errors": {
    "static_ips": [
      "static ips per apn per subscriber must be unique. eg a
subscriber may not be assigned static ip 100.64.1.1 for internet
and also 100.64.1.2 for internet"
    ]
  }
}
```

### See Also:

- [Create Subscriber](#) - Subscriber provisioning
- [Update Subscriber](#) - Modifying subscriber configuration
- [Complete Static IP Provisioning Example](#) - End-to-end workflow

---

## Roaming Management

Roaming profiles control whether subscribers can access data and IMS services on visited networks. Profiles are assigned to [subscribers](#) and consist of rules matched by MCC/MNC.

### List Roaming Profiles

**Endpoint:** `GET /api/roaming/profile`

# Create Roaming Profile

**Endpoint:** POST /api/roaming/profile

**Request Body:**

```
{
  "roaming_profile": {
    "name": "US Carriers Only",
    "data_action_if_no_rules_match": "deny",
    "ims_action_if_no_rules_match": "deny",
    "roaming_rules": []
  }
}
```

**Action Values:**

- "allow" - Allow
- "deny" - Deny

**Default Actions:**

- data\_action\_if\_no\_rules\_match - Action when no **roaming rule** matches
- ims\_action\_if\_no\_rules\_match - IMS-specific default action

# List Roaming Rules

**Endpoint:** GET /api/roaming/rule

# Create Roaming Rule

**Endpoint:** POST /api/roaming/rule

**Request Body:**

```
{
  "roaming_rule": {
    "name": "Allow AT&T",
    "mcc": "310",
    "mnc": "410",
    "data_action": "allow",
    "ims_action": "allow"
  }
}
```

### Fields:

- `mcc` - Mobile Country Code (3 digits)
- `mnc` - Mobile Network Code (2-3 digits)
- `data_action` - "allow" or "deny" data services
- `ims_action` - "allow" or "deny" IMS/voice services

### See Also:

- [Roaming Documentation](#) - Detailed configuration and examples
- [Protocol Flows](#) - How roaming control works in Diameter flows

---

## EIR Management

OmniHSS functions as an Equipment Identity Register (EIR) via the S13 Diameter interface. EIR rules control device access based on IMEI patterns.

See [EIR Documentation](#) for detailed equipment identity checking, S13 interface flows, and IMEI validation.

### List EIR Rules

**Endpoint:** `GET /api/eir/rule`

# Create EIR Rule

**Endpoint:** `POST /api/eir/rule`

## Request Body:

```
{
  "eir_rule": {
    "name": "Block iPhone 6",
    "imei_regex": "^35[0-9]{6}0[0-9]{7}$",
    "action": 1
  }
}
```

## Fields:

- `name` - Descriptive name for the rule
- `imei_regex` - Regular expression to match IMEI numbers
- `action` - Whitelist (0), Blacklist (1), or Greylist (2)

## Action Values:

- `0` - Whitelist (allow)
- `1` - Blacklist (deny)
- `2` - Greylist (allow but track)

## Use Cases:

- Block stolen devices (blacklist specific IMEIs)
- Restrict device types (blacklist by TAC pattern)
- Allow only approved devices (whitelist pattern with deny-all default)

## See Also:

- [Protocol Flows](#) - S13 interface and EIR check flow
  - [Architecture Overview](#) - OmniHSS EIR function
-

# Additional Documentation

For more information, see the following documentation:

- **Status and Health** - API health check endpoints
- **Error Handling** - Common errors and troubleshooting
- **API Usage Examples** - Complete provisioning workflows

---

[← Back to Operations Guide](#) | [Next: Control Panel →](#)

# API Status and Health

[← Back to API Reference](#)

---

## System Status

Check if the API is responding.

**Endpoint:** `GET /api/status`

**Example Request:**

```
curl -k https://hss.example.com:8443/api/status
```

**Example Response:**

```
{  
  "status": "ok"  
}
```

**Use Case:** Health check for load balancers and monitoring systems.

---

[← Back to API Reference](#)

# OmniHSS Architecture Overview

[← Back to Operations Guide](#)

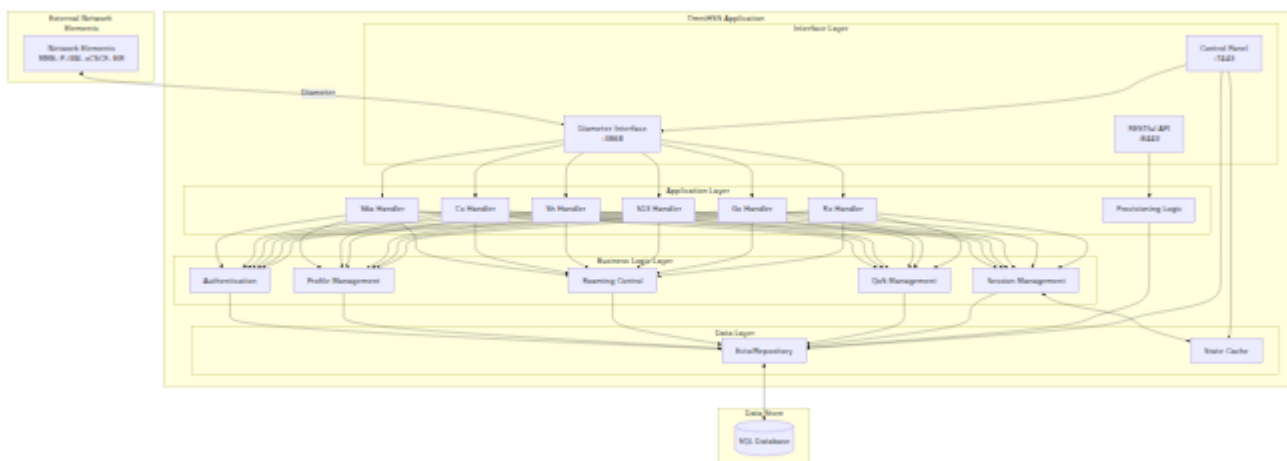
---

## Table of Contents

- [System Overview](#)
  - [Component Architecture](#)
  - [Diameter Stack](#)
  - [Application Layer](#)
  - [Data Layer](#)
  - [External Interfaces](#)
  - [Deployment Architecture](#)
- 

## System Overview

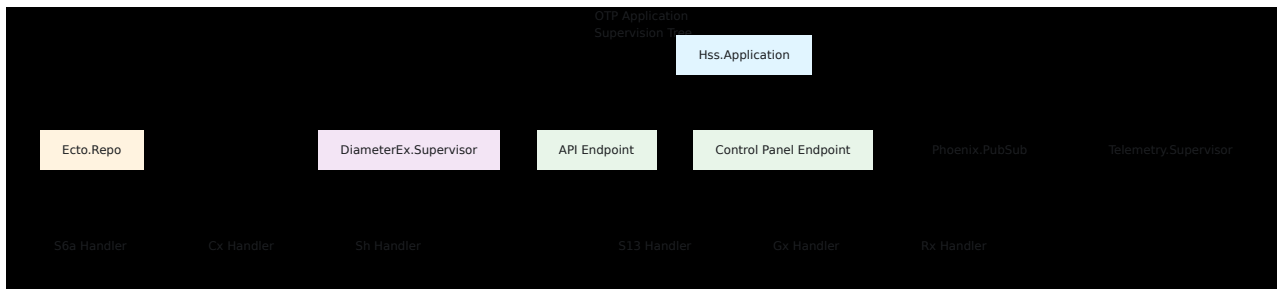
OmniHSS is built on Elixir and the Erlang/OTP platform, providing a highly concurrent, fault-tolerant system designed for telecommunications workloads. The architecture follows a layered approach with clear separation of concerns.



---

# Component Architecture

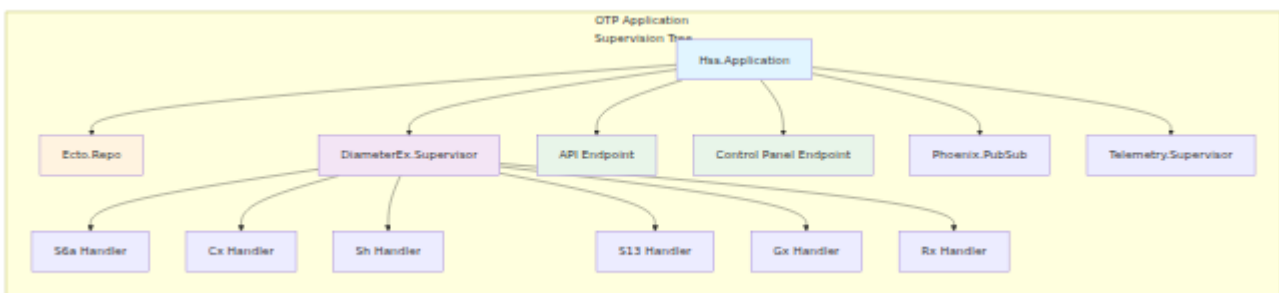
## Core Components



## Diameter Application Handlers

Each Diameter application (S6a, Cx, Sh, S13, Gx, Rx) is implemented as a DiameterEx handler module that:

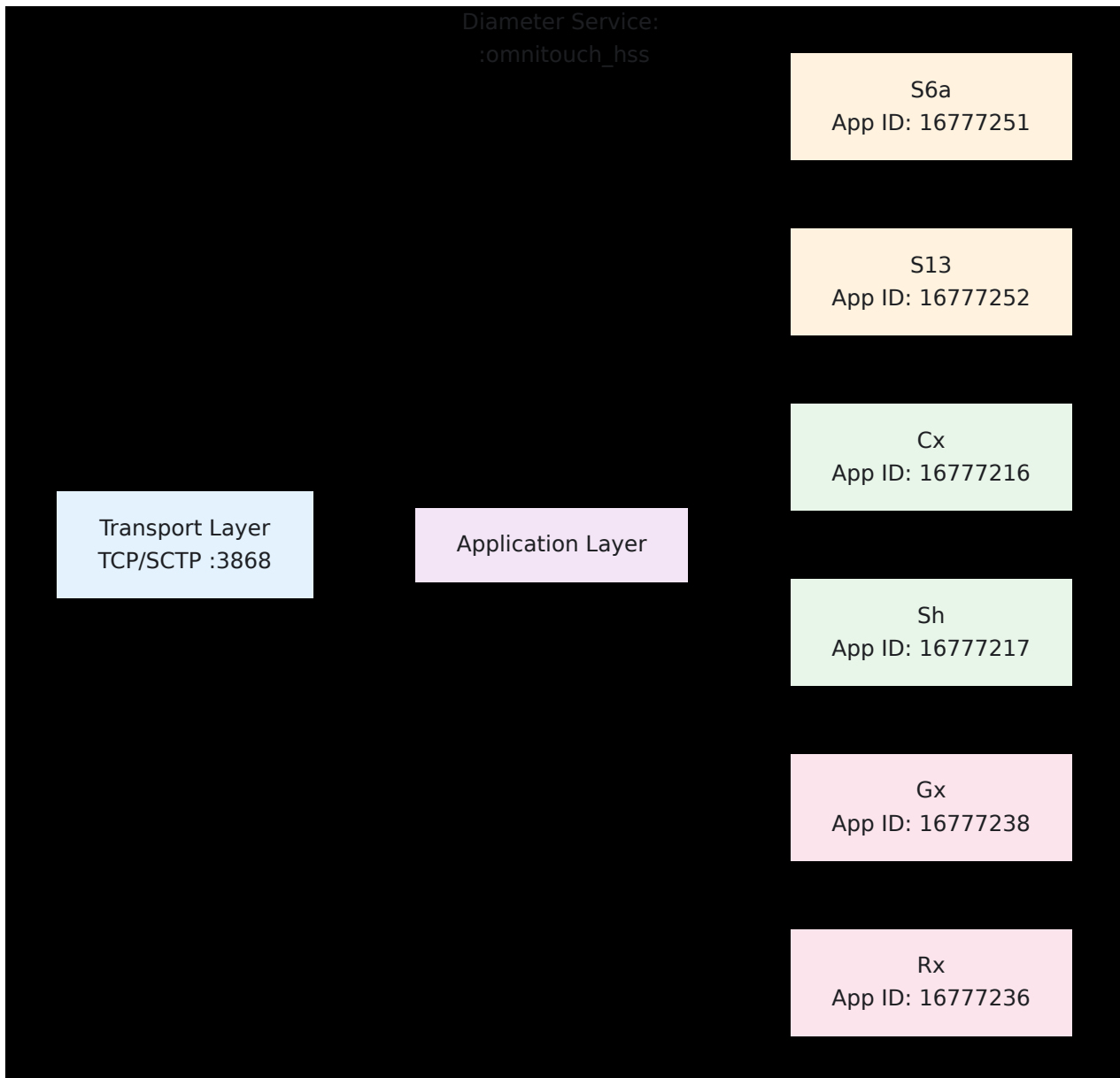
1. **Registers with DiameterEx** - Subscribes to specific Diameter Application IDs
2. **Validates Requests** - Extracts AVPs, validates subscriber state
3. **Processes Business Logic** - Calls appropriate business logic modules
4. **Constructs Responses** - Builds Diameter answer messages with AVPs
5. **Handles Errors** - Returns appropriate Diameter result codes



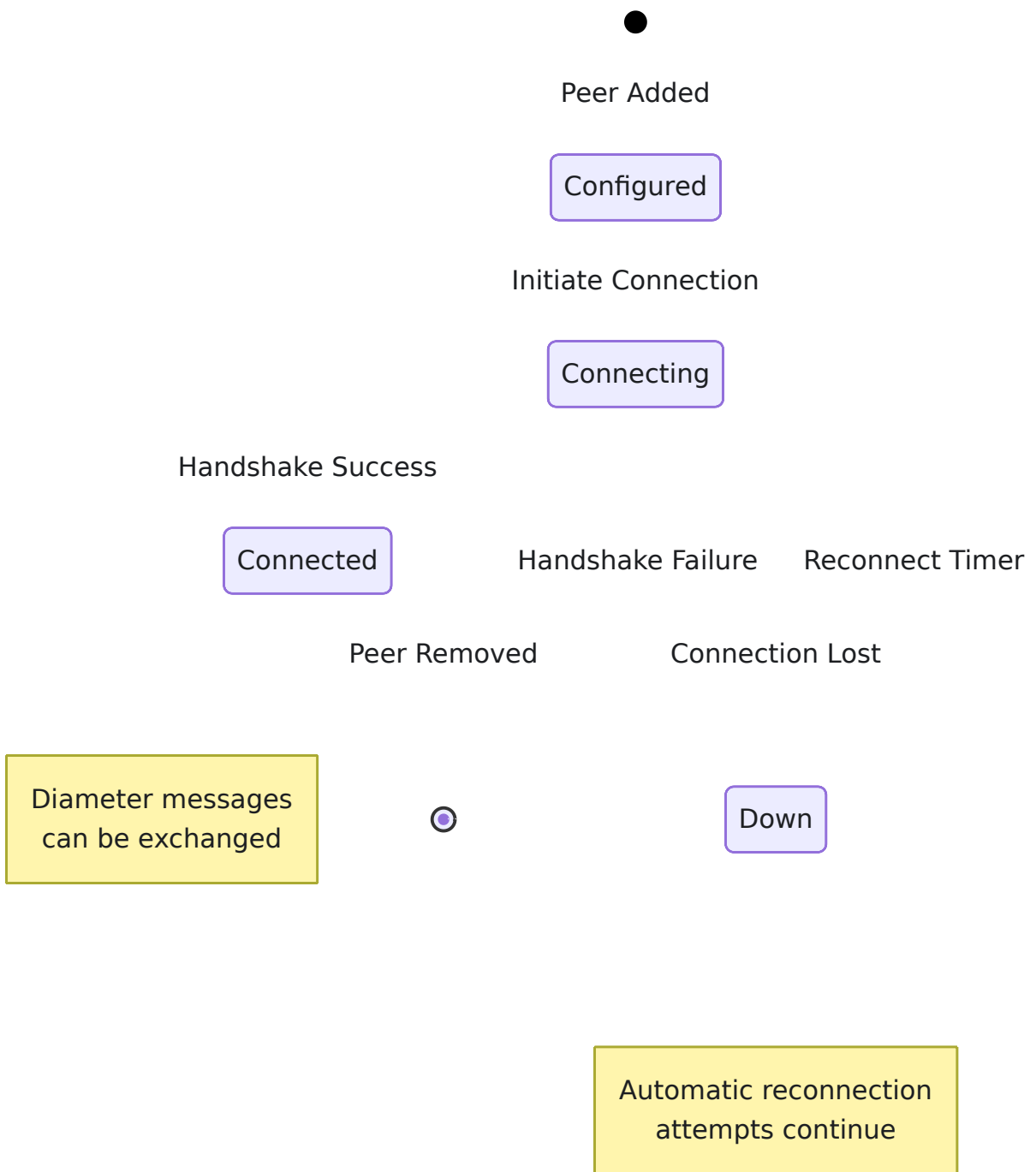
# Diameter Stack

## Diameter Service Configuration

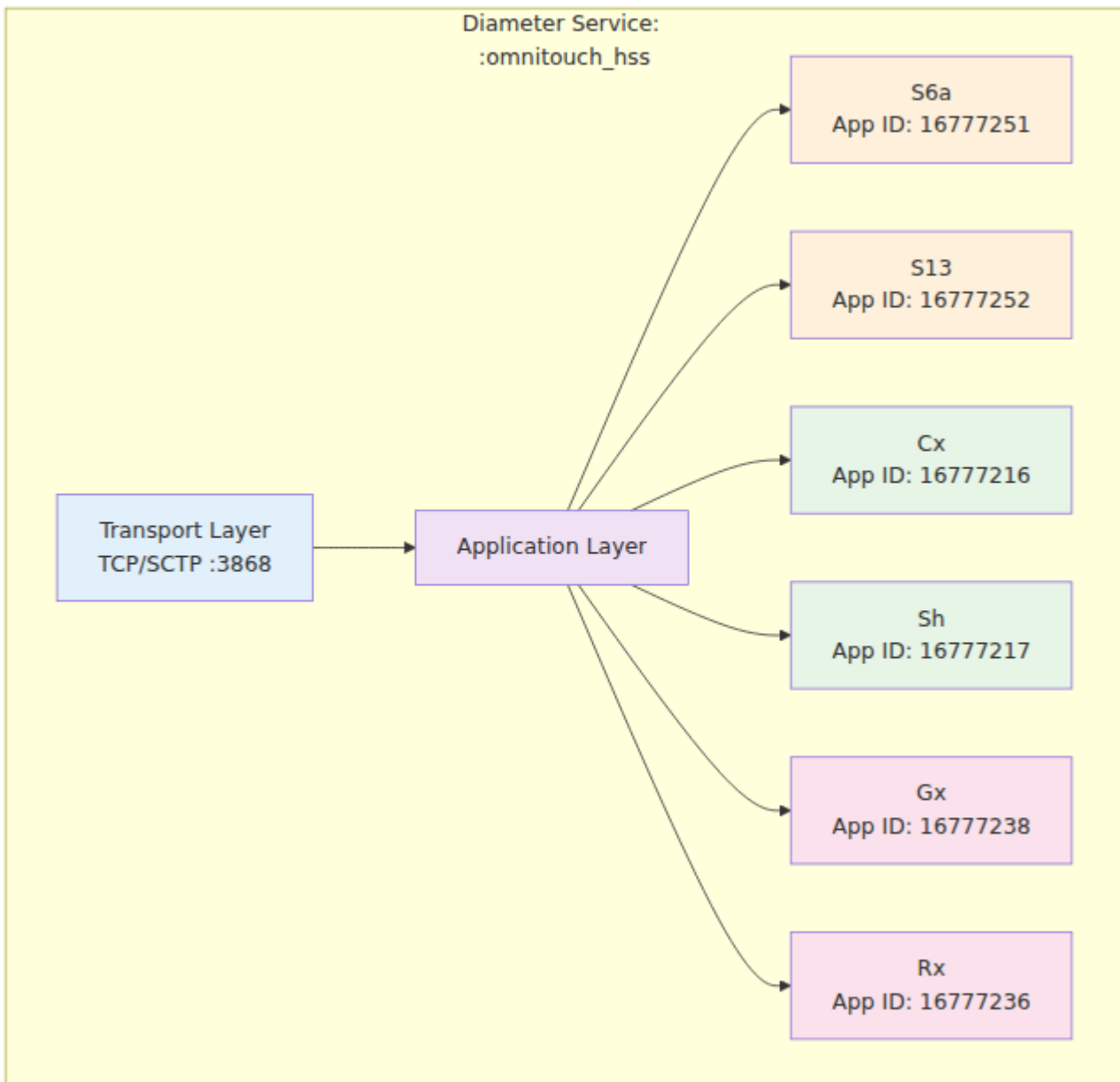
OmniHSS configures a single Diameter service with multiple supported applications:



# Peer Connection Management



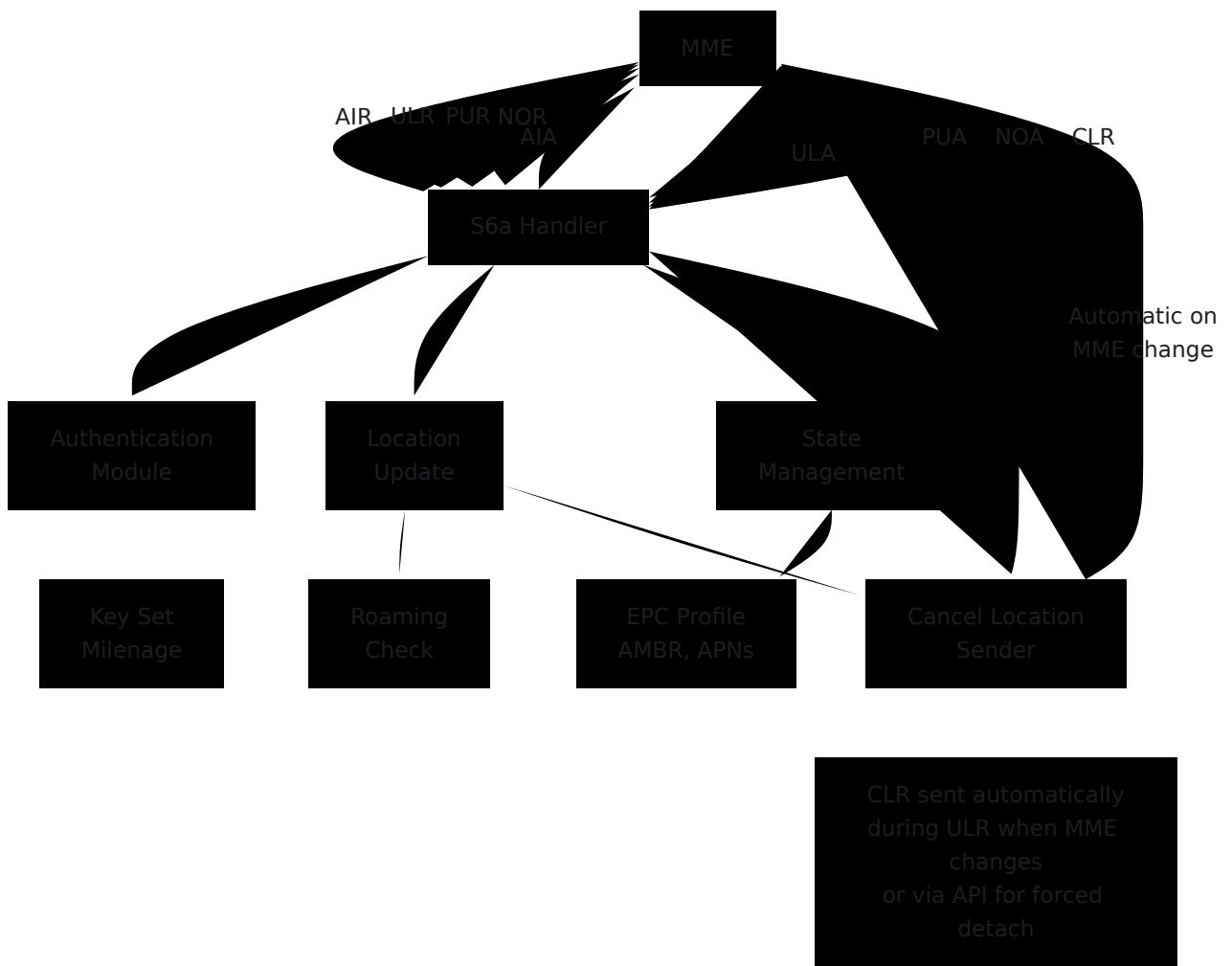
# Diameter Message Flow



## Application Layer

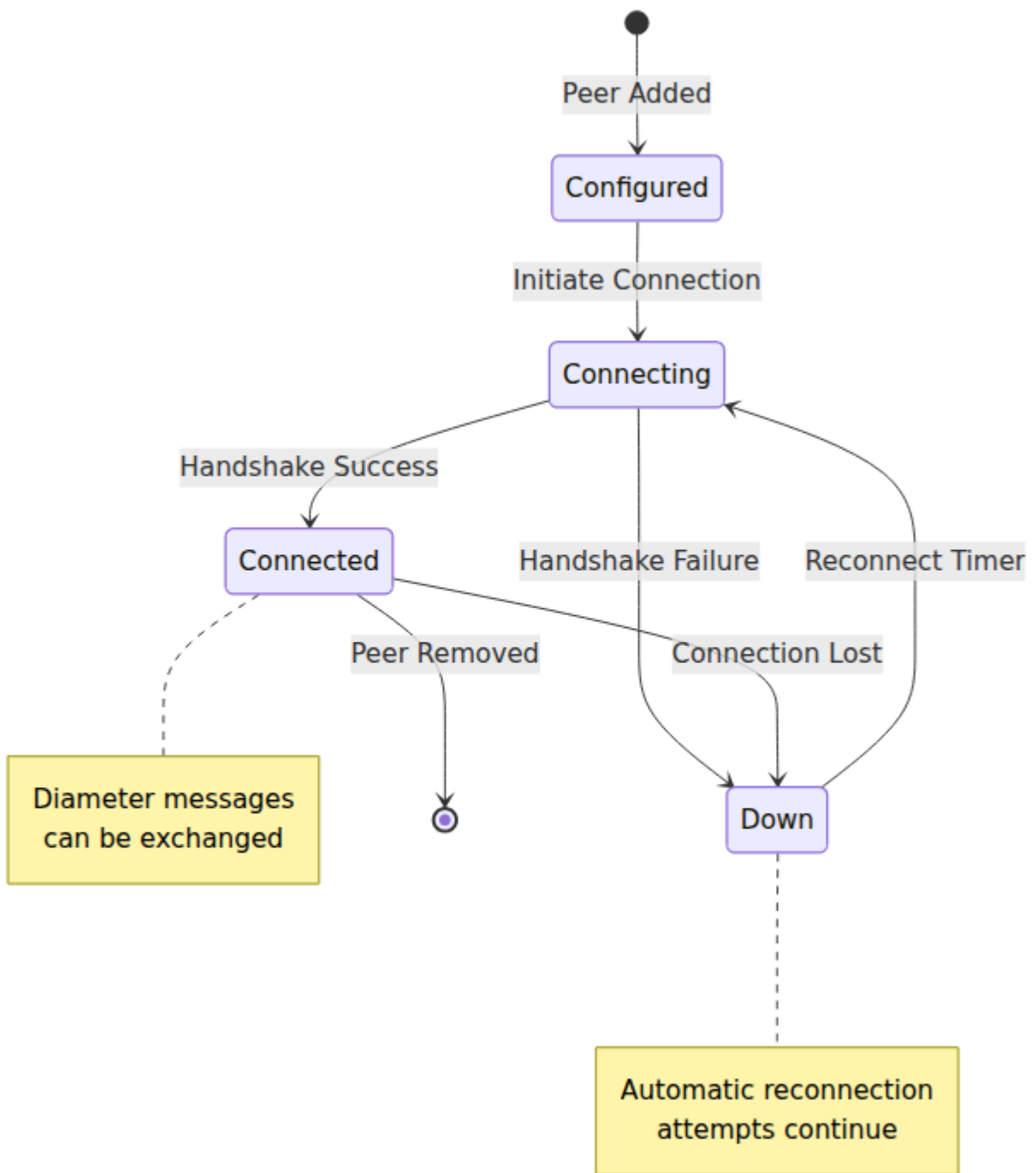
### S6a Interface (LTE/EPC)

Handles authentication and mobility management for LTE networks.



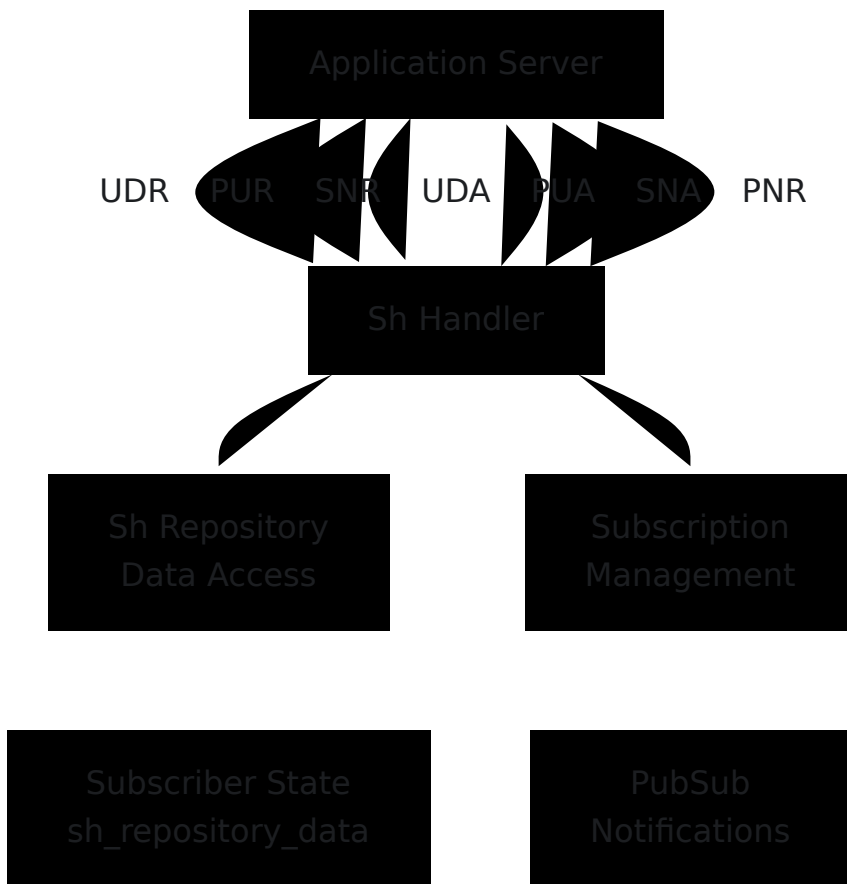
## Cx Interface (IMS)

Handles IMS registration and authentication.



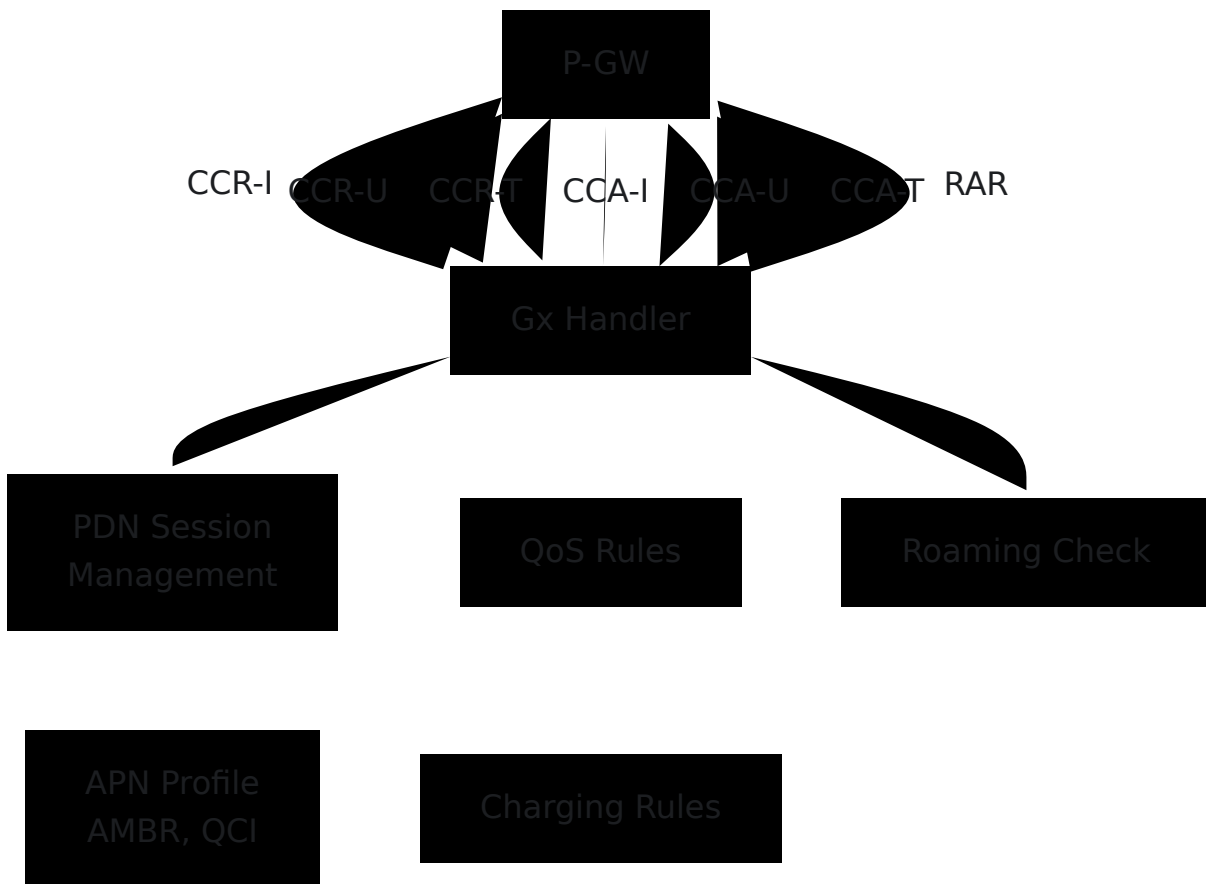
## Sh Interface (IMS Profile Data)

Provides IMS application servers access to subscriber profile data.



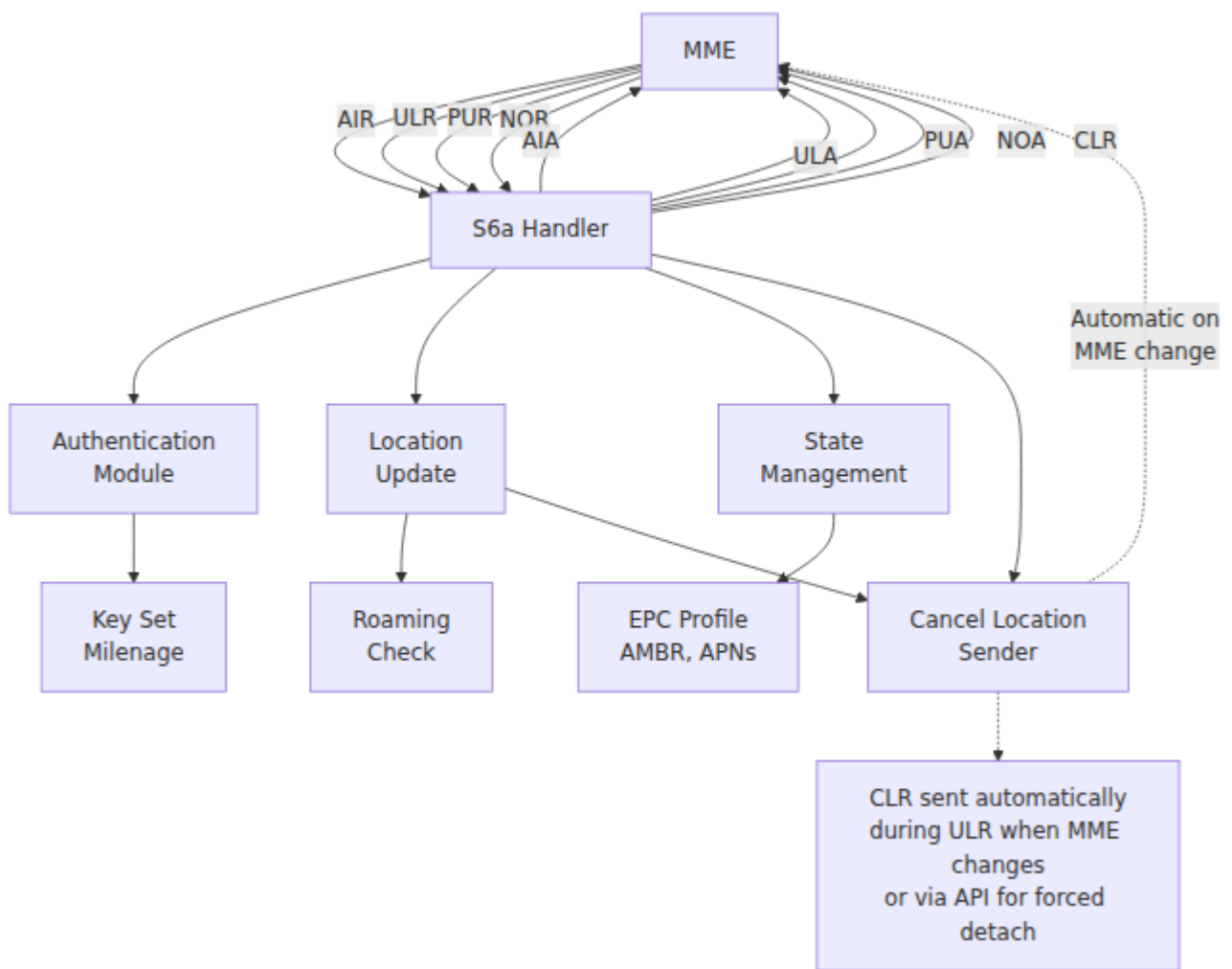
## Gx Interface (Policy Control)

Manages policy and charging control for data sessions. **See [PCRF Documentation](#) for details.**



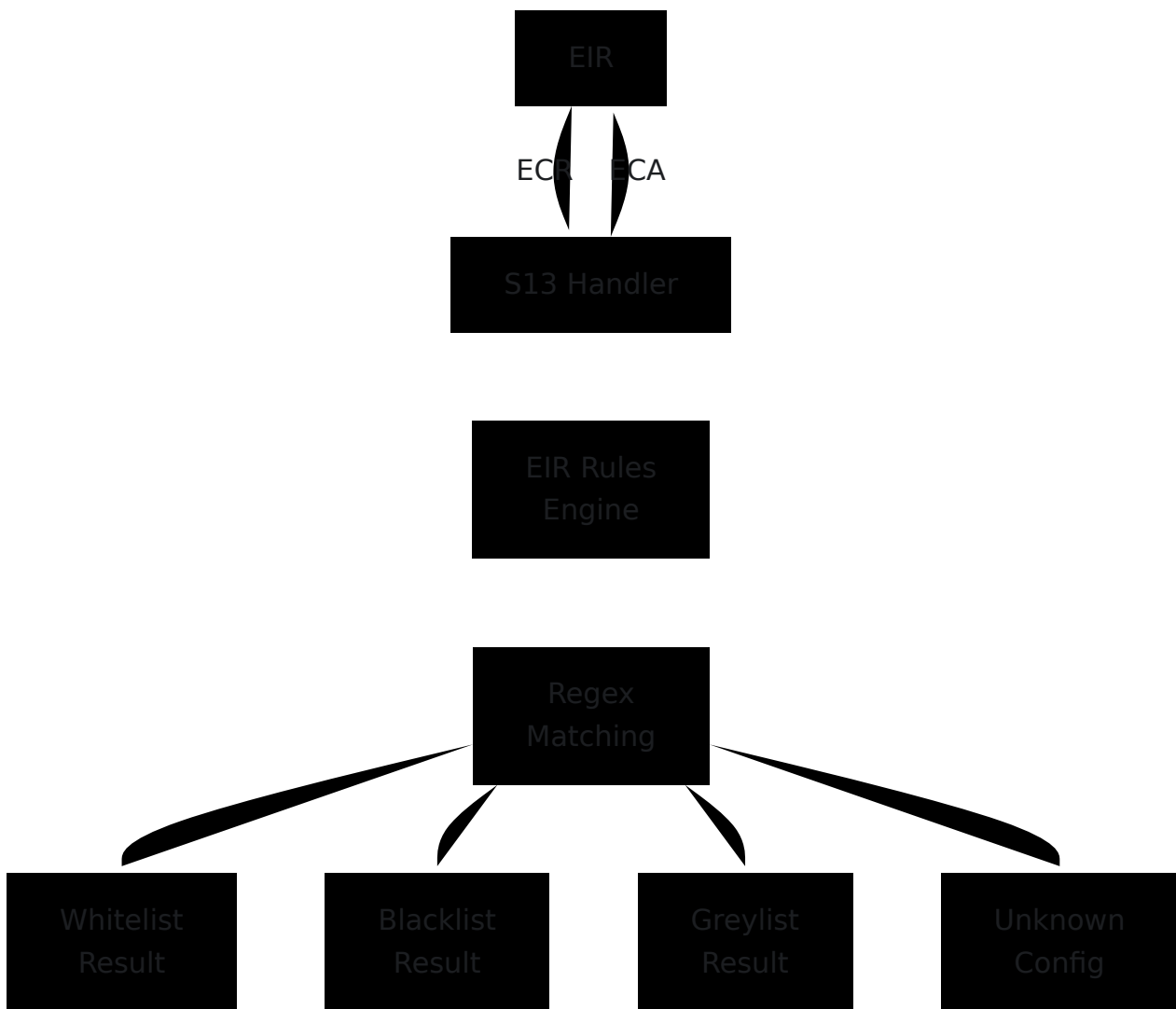
## Rx Interface (IMS Media)

Controls IMS media policy and dedicated bearers for VoLTE. **See [PCRF Documentation](#) for details.**



## S13 Interface (EIR)

Validates device IMEI against equipment identity rules. **See [EIR Documentation](#) for details.**



---

## Data Layer

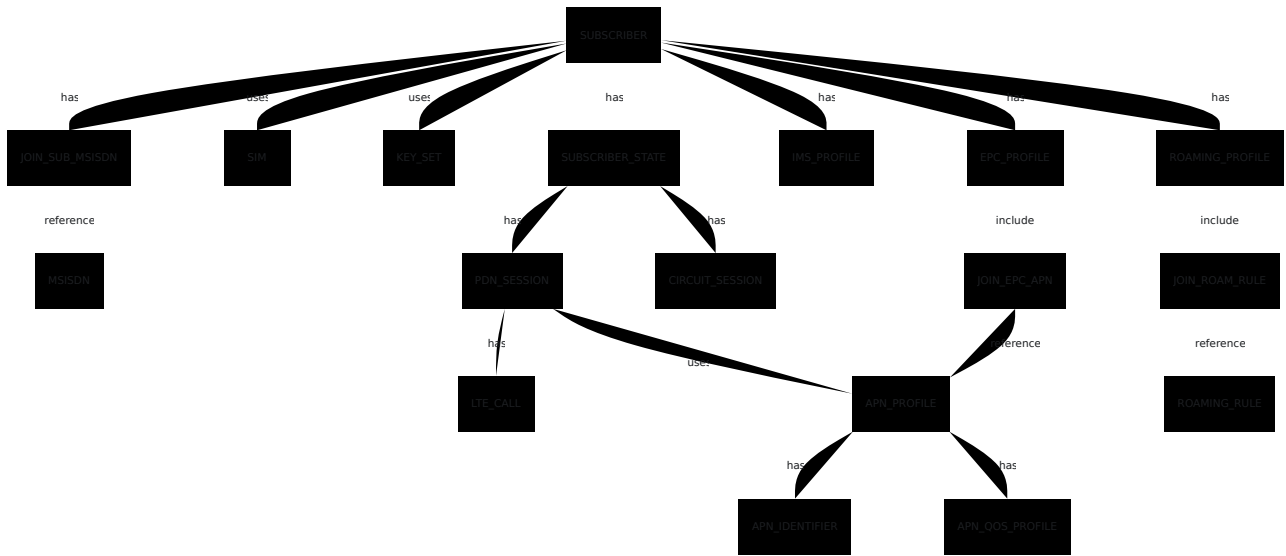
### Database Backend

OmniHSS uses **PostgreSQL** as its database backend and **Ecto** as its database abstraction layer. All tables use UUID primary keys for globally unique identification across sites.

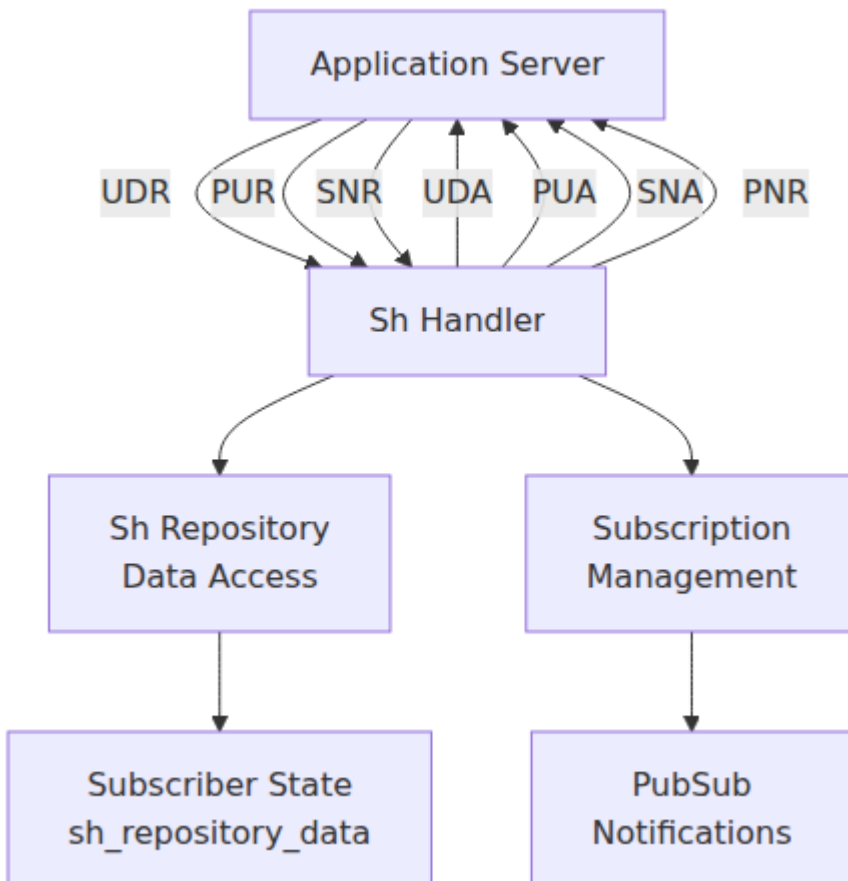
For multi-site deployments, OmniHSS uses PostgreSQL's bidirectional logical replication. Each site operates as an independent read/write primary with asynchronous WAL-based replication between sites.

See [Database Replication](#) for multi-site replication configuration and operations.

## Database Schema Overview

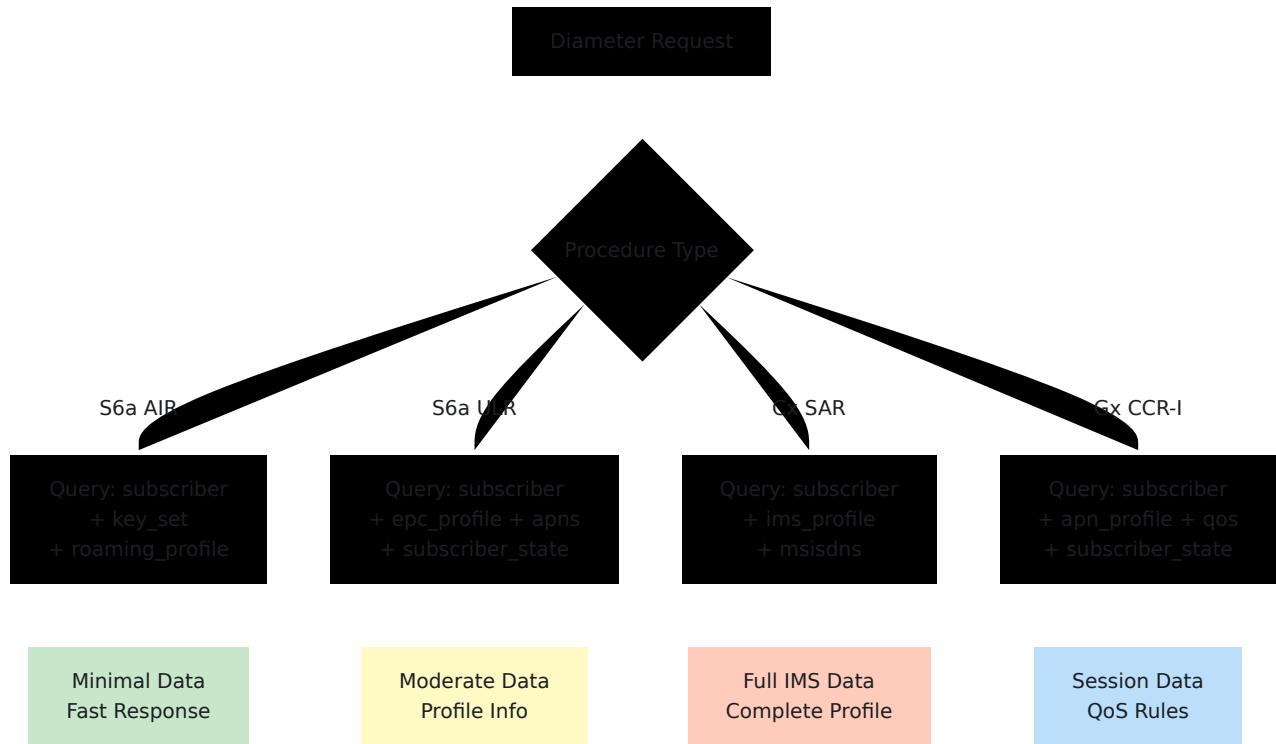


## Ecto Repository Pattern



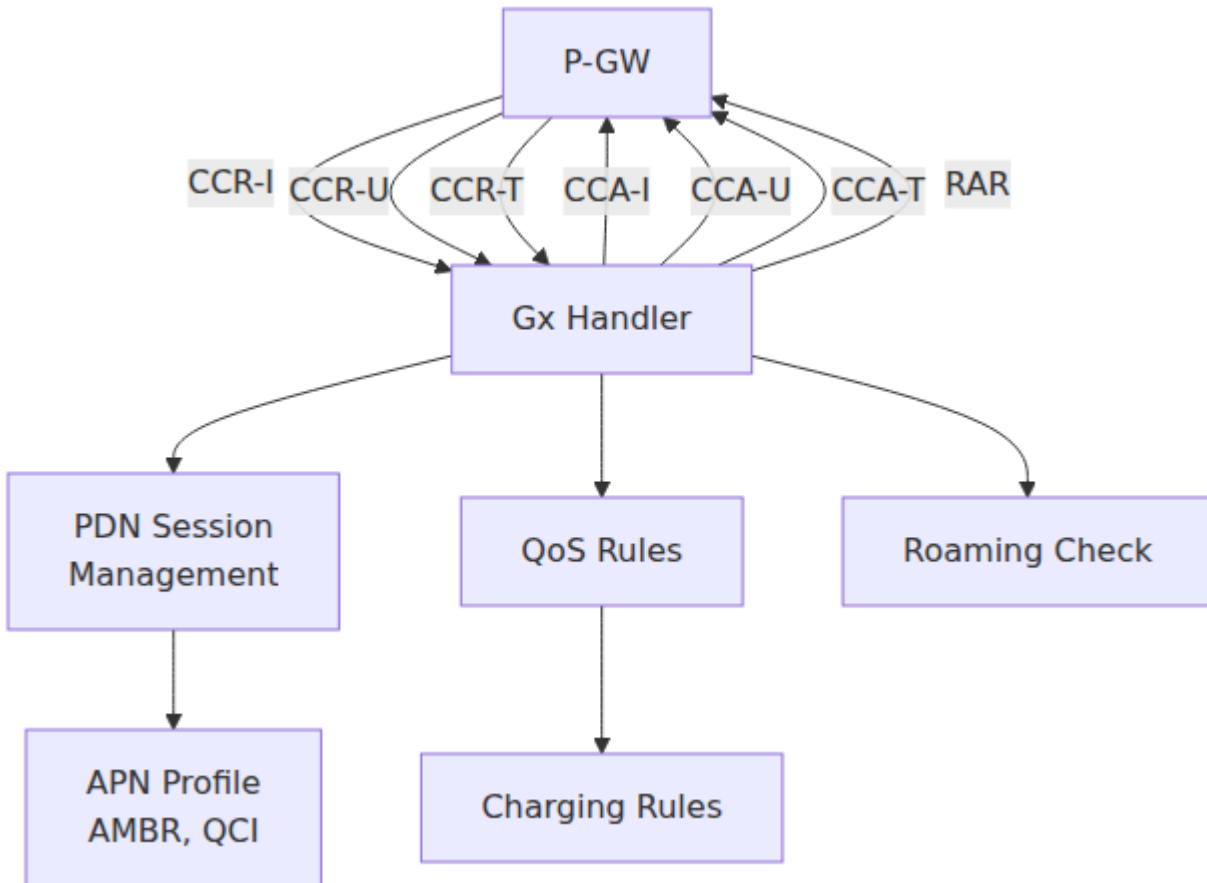
# Optimized Query Strategy

Each Diameter procedure uses optimized queries that preload only necessary associations:

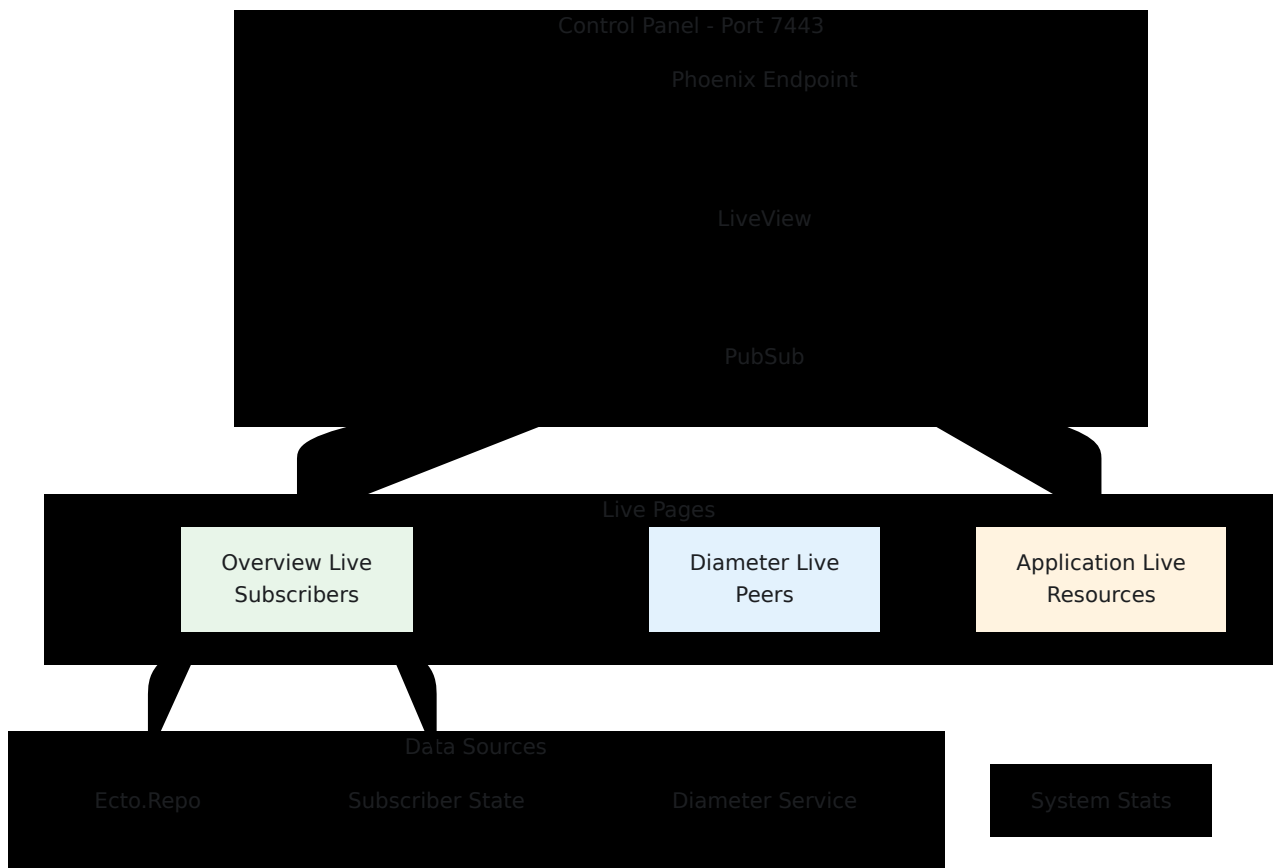


# External Interfaces

## API Architecture

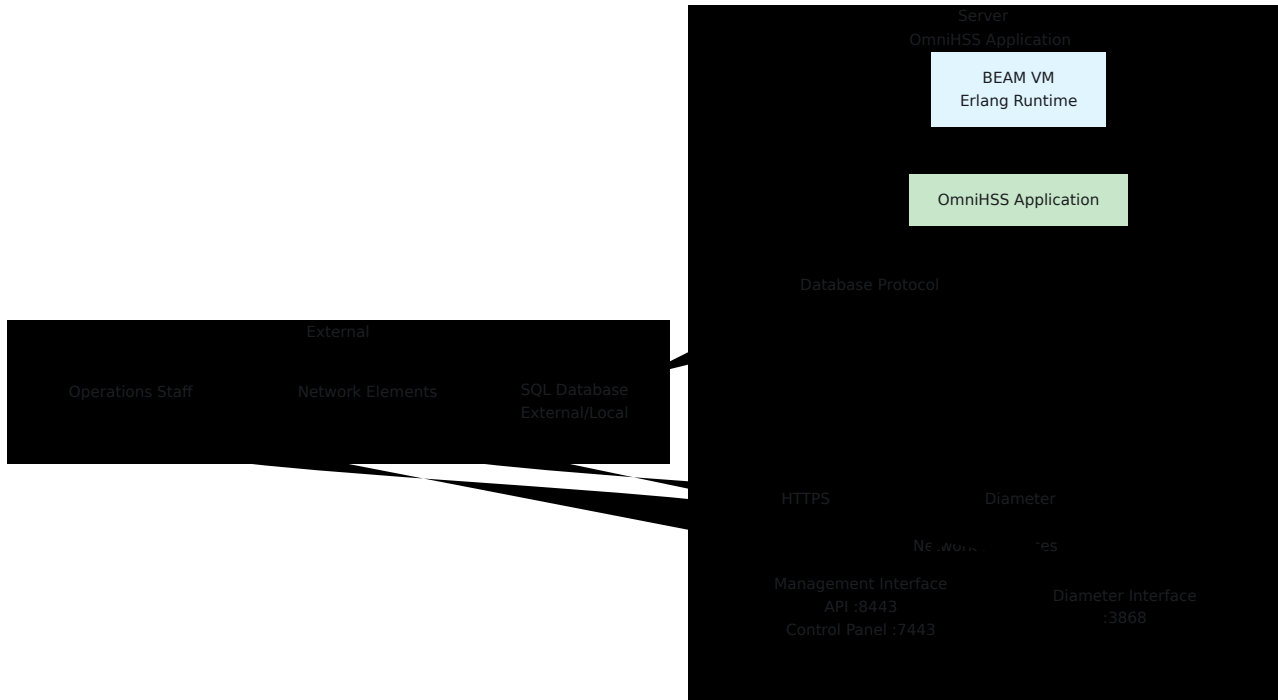


# Control Panel Architecture



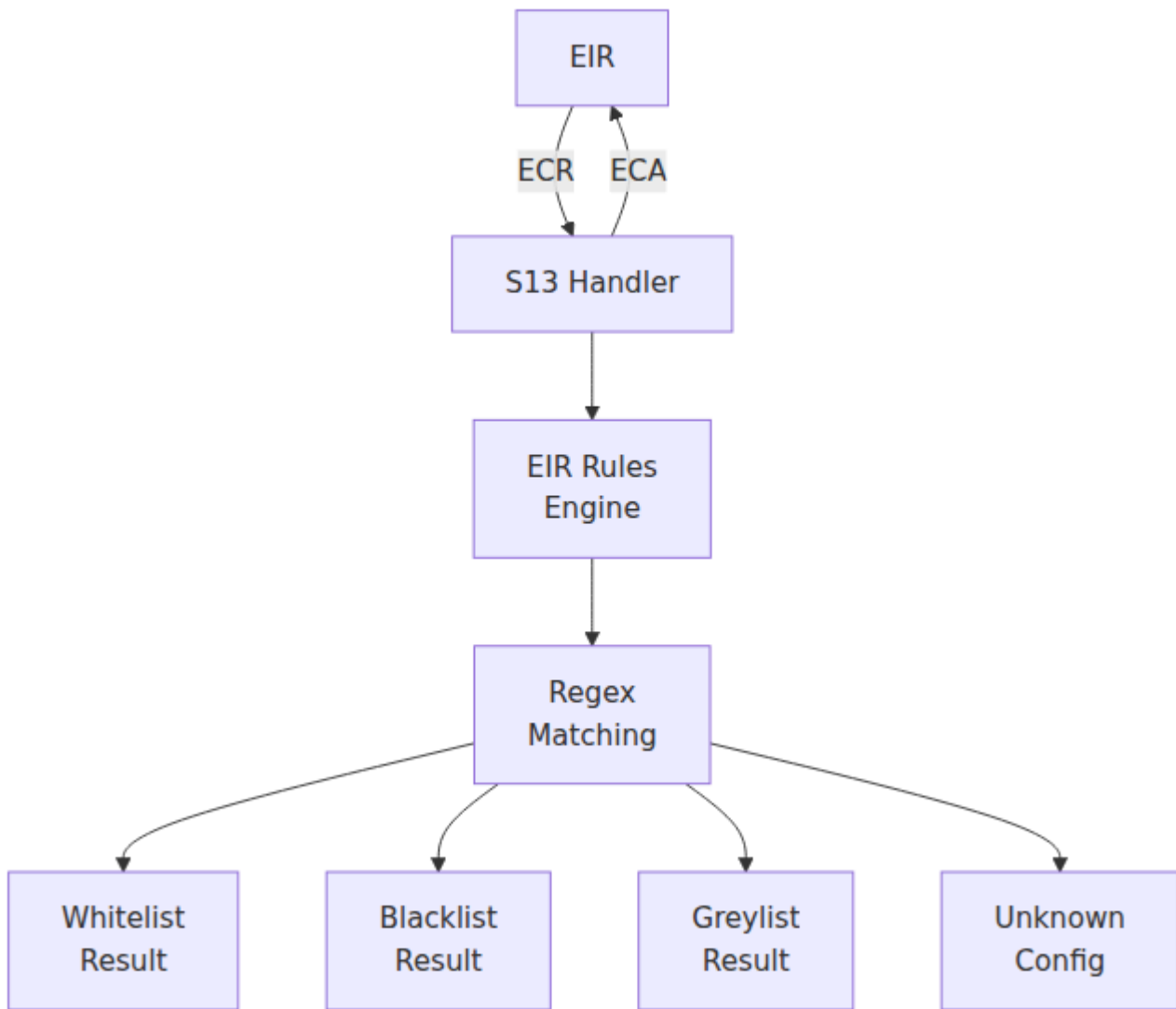
# Deployment Architecture

## Single Node Deployment



## Multi-Site HA Deployment (Logical Replication)

For high-availability deployments across geographic sites, OmniHSS uses PostgreSQL bidirectional logical replication. Each site has an independent read/write database.



### Key Characteristics:

- **Both Sites Read/Write:** Local writes are instant, never blocked by remote site
- **Asynchronous Replication:** Changes stream via WAL — no write latency penalty from replication
- **Partition Tolerant:** Both sites continue operating if the inter-site link goes down. WAL queues locally and catches up on restore
- **No Quorum:** No minimum node count required. Works identically with one site or many
- **UUID Primary Keys:** Globally unique IDs eliminate conflicts between sites

See [Database Replication](#) for detailed configuration and operations.

---

# Process Flow Example: Authentication

This example shows the complete flow for an authentication request:



## Key Architectural Principles

### 1. Fault Tolerance

- Erlang/OTP supervision trees automatically restart failed processes
- Isolated Diameter handlers prevent cascading failures
- Database connection pooling with automatic reconnection

### 2. Concurrency

- Each Diameter request handled in its own process
- No shared state between request handlers
- Database connection pooling for parallel queries

### **3. Modularity**

- Each Diameter application in separate module
- Clear separation between interface, business logic, and data layers
- Pluggable authentication algorithms

### **4. Performance**

- Optimized database queries with selective preloading
- Minimal data transfer for each procedure type
- Connection pooling and keepalive

### **5. Observability**

- Real-time monitoring via Control Panel
- Structured logging throughout application
- Diameter peer status tracking
- Subscriber state tracking with timestamps

---

[← Back to Operations Guide](#) | [Next: Configuration](#) →

# OmniHSS Configuration Guide

[← Back to Operations Guide](#)

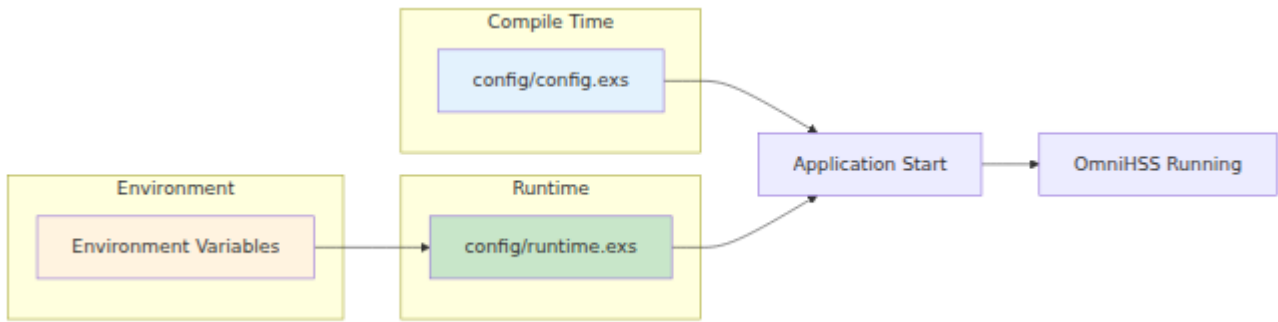
---

## Table of Contents

- [Configuration File Overview](#)
  - [License Client Configuration](#)
  - [Runtime Configuration](#)
  - [Database Configuration](#)
  - [Diameter Configuration](#)
  - [Network Configuration](#)
    - [Home PLMN Configuration](#)
    - [HSS Core Configuration](#)
  - [IMS Configuration](#)
  - [EIR Configuration](#)
  - [API and Control Panel Configuration](#)
  - [Configuration Workflow](#)
- 

## Configuration File Overview

OmniHSS uses two primary configuration files:



## **config/config.exs (Compile Time)**

Contains static configuration that doesn't change between environments:

- Control Panel page configuration
- API endpoint configuration
- Telemetry settings

## **config/runtime.exs (Runtime)**

Contains environment-specific configuration that changes per deployment:

- Database connection parameters
- Diameter peer configuration
- Home PLMN settings
- IMS S-CSCF selection
- Network interface bindings

---

# **License Client Configuration**

The License Client validates the HSS license with a remote license server:

```
# config/runtime.exs

config :license_client,
  # License server API endpoints (list for failover)
  license_server_api_urls:
  ["https://license.example.com:8443/api"],

  # Licensed organization name
  licensee: "Your Organization Name",

  # Product identifier
  product_name: "omnihss"
```

### License Configuration Parameters:

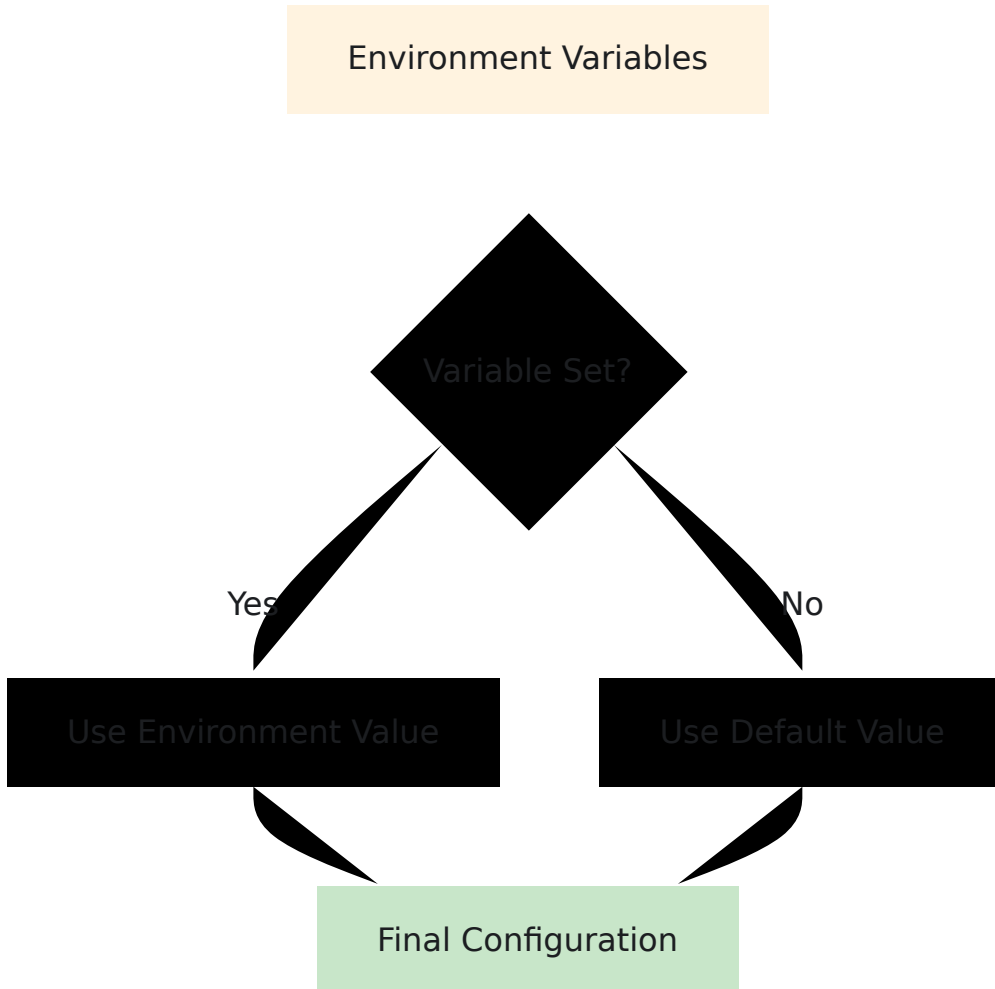
Parameter	Description	Required	Example
<code>license_server_api_urls</code>	List of license server URLs	Yes	<code>["https://10.0.0.</code>
<code>licensee</code>	Organization name on license	Yes	<code>"ACME Telecom"</code>
<code>product_name</code>	Product identifier for license	Yes	<code>"omnihss"</code>

### Important Notes:

- License server must be reachable from HSS
  - Use HTTPS for secure license validation
  - Multiple URLs provide failover capability
  - License validation occurs at startup and periodically
-

# Runtime Configuration

## Configuration Priority



## Environment Variable Pattern

OmniHSS follows this pattern for configuration:

- Environment variable names are UPPERCASE with underscores
  - Default values are provided in runtime.exs
  - Database credentials should use environment variables in production
-

# Database Configuration

## Basic Database Configuration

```
# config/runtime.exs

config :hss, Hss.Repo,
  # Database connection parameters
  username: System.get_env("DATABASE_USERNAME", "root"),
  password: System.get_env("DATABASE_PASSWORD", "password"),
  hostname: System.get_env("DATABASE_HOSTNAME", "localhost"),
  database: System.get_env("DATABASE_NAME", "omnihss"),

  # Connection pool settings
  pool_size:
String.to_integer(System.get_env("DATABASE_POOL_SIZE", "20")),

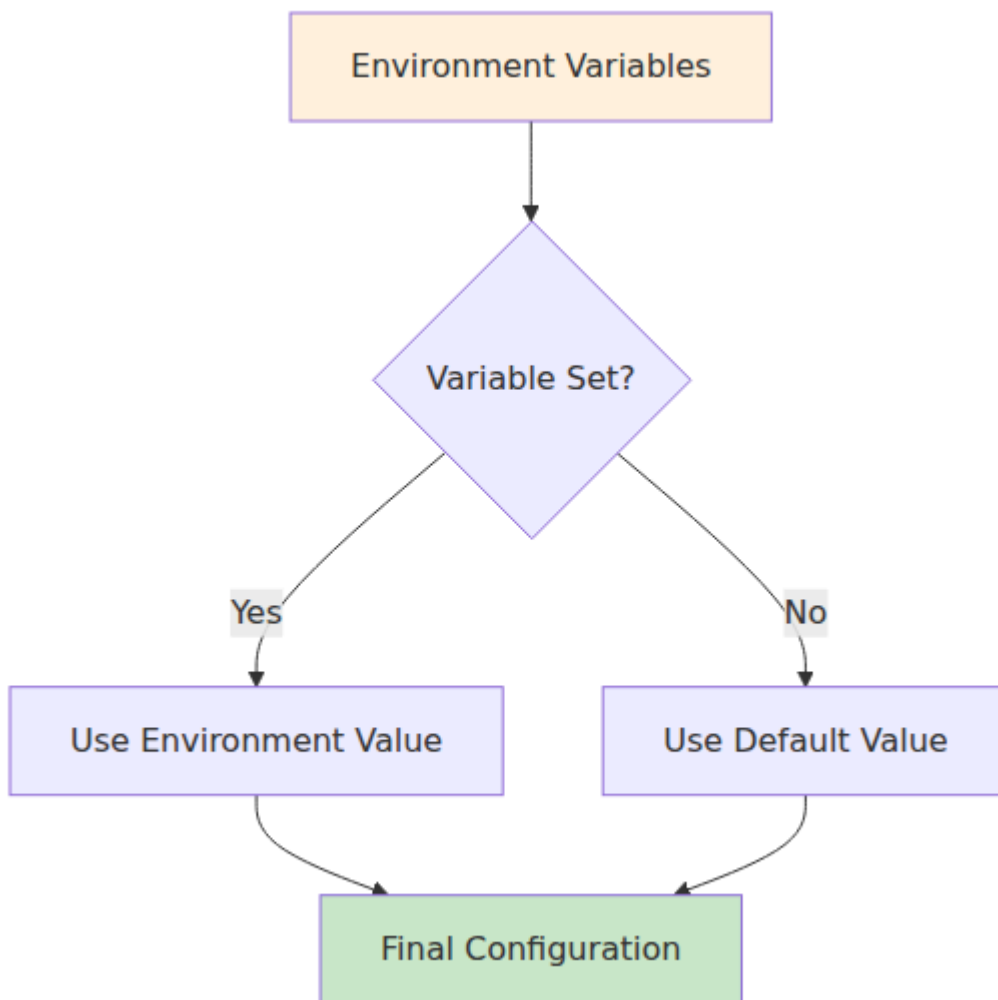
  # Timeouts (in milliseconds)
  timeout: 15_000,
  connect_timeout: 15_000,

  # Additional options
  show_sensitive_data_on_connection_error: false
```

# Database Configuration Parameters

Parameter	Description	Default	Recommendation
<code>username</code>	SQL Database username	<code>"root"</code>	Use dedicated user in production
<code>password</code>	SQL Database password	<code>"password"</code>	Use strong password, store in env var
<code>hostname</code>	SQL Database server hostname	<code>"localhost"</code>	Use FQDN or IP in production
<code>database</code>	Database name	<code>"omnihss"</code>	Keep default unless multiple instances
<code>pool_size</code>	Connection pool size	<code>20</code>	Adjust based on load (10-50 typical)

# Pool Size Tuning



## Guidelines:

- Start with 20 connections
- Monitor for "connection pool timeout" errors
- Increase by 10 if timeouts occur under normal load
- Each connection uses ~4MB of memory
- Too many connections can degrade SQL Database performance

## Example: Production Database Configuration

```
# config/runtime.exs - Production example

config :hss, Hss.Repo,
  username: System.fetch_env!("DATABASE_USERNAME"),      #
  Required in production
  password: System.fetch_env!("DATABASE_PASSWORD"),      #
  Required in production
  hostname: System.get_env("DATABASE_HOSTNAME",
  "db.internal.example.com"),
  database: System.get_env("DATABASE_NAME", "omnihss"),
  port: String.to_integer(System.get_env("DATABASE_PORT",
  "5432")),
  pool_size:
String.to_integer(System.get_env("DATABASE_POOL_SIZE", "30")),
  ssl: true,
  ssl_opts: [
    cacertfile: "/etc/ssl/certs/postgres-ca.pem",
    verify: :verify_peer
  ]
]
```

---

# Diameter Configuration

## Diameter Service Configuration

```
# config/runtime.exs

diameter_config = %{
  service_name: :omnitouch_hss,

  # Network binding
  listen_ip: System.get_env("DIAMETER_LISTEN_IP", "10.7.25.186"),
  listen_port:
String.to_integer(System.get_env("DIAMETER_LISTEN_PORT", "3868")),

  # Diameter identity
  host: System.get_env("DIAMETER_HOST", "omnihss"),
  realm: System.get_env("DIAMETER_REALM",
"epc.mnc001.mcc001.3gppnetwork.org"),

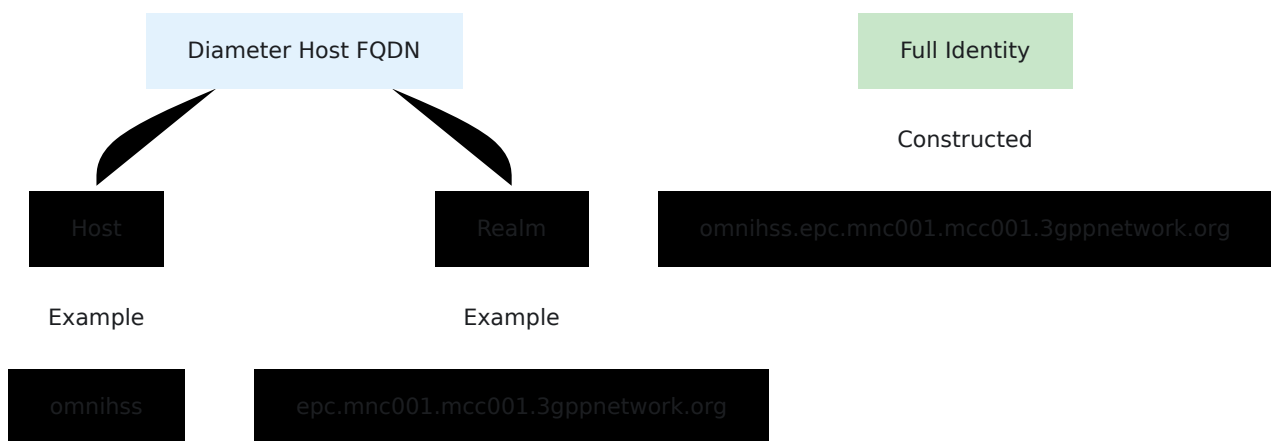
  # Product identification
  product_name: "OmniHSS",
  vendor_id: 10415, # 3GPP
  supported_vendor_ids: [5535, 10415],

  # Protocol settings
  request_timeout: 5000,

  # Peer configuration
  peers: [
    # Add peer configurations here
  ]
}

config :hss, :diameter, diameter_config
```

# Diameter Identity Configuration



## Guidelines:

- **Host:** Short hostname of the HSS (e.g., "omnihss", "hss01")
- **Realm:** Diameter realm matching your PLMN (e.g., "epc.mnc001.mcc001.3gppnetwork.org")
- **Full Identity:** Constructed as `{host}.{realm}`

## Adding Diameter Peers

### Static Peer Configuration (Connect Mode)

```
# config/runtime.exs

peers: [
  # MME Peer Example
  %{
    host: "mme01.epc.mnc001.mcc001.3gppnetwork.org",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    ip: "10.7.25.100",
    port: 3868,
    transport: :sctp, # or :tcp
    applications: [:s6a]
  },

  # P-GW Peer Example
  %{
    host: "pgw01.epc.mnc001.mcc001.3gppnetwork.org",
    realm: "epc.mnc001.mcc001.3gppnetwork.org",
    ip: "10.7.25.101",
    port: 3868,
    transport: :sctp,
    applications: [:gx]
  },

  # I-CSCF Peer Example
  %{
    host: "icscf01.ims.mnc001.mcc001.3gppnetwork.org",
    realm: "ims.mnc001.mcc001.3gppnetwork.org",
    ip: "10.7.25.102",
    port: 3868,
    transport: :tcp,
    applications: [:cx]
  }
]
```

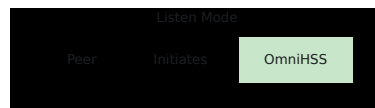
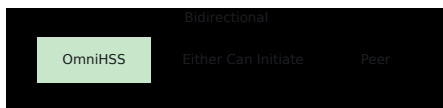
## Listen-Only Mode

For environments where peers initiate connections to the HSS:

```
# config/runtime.exs

diameter_config = %{
  # ... other config ...
  peers: [] # Empty - accept incoming connections only
}
```

## Diameter Peer Connection Modes



## Transport Protocol Selection

Transport	Advantages	Disadvantages	Recommendation
<b>SCTP</b>	Multi-streaming, better failure detection	Requires kernel support, firewall config	Preferred for Diameter
<b>TCP</b>	Universal support, simpler firewall rules	Single stream, slower failure detection	Use if SCTP unavailable

# Network Configuration

## Home PLMN Configuration

The home PLMN identifies your network operator:

```
# config/runtime.exs

config :hss, :home_plmn, %{
  mcc: System.get_env("HOME_PLMN_MCC", "001"), # Mobile Country
Code
  mnc: System.get_env("HOME_PLMN_MNC", "001") # Mobile Network
Code
}
```

## HSS Core Configuration

These settings control HSS behavior and features:

```
# config/runtime.exs

config :hss,
  # Ecto repositories for database operations
  ecto_repos: [Hss.Repo],

  # CLR (Cancel Location Request) on MME change
  send_clr_on_mme_change: true,

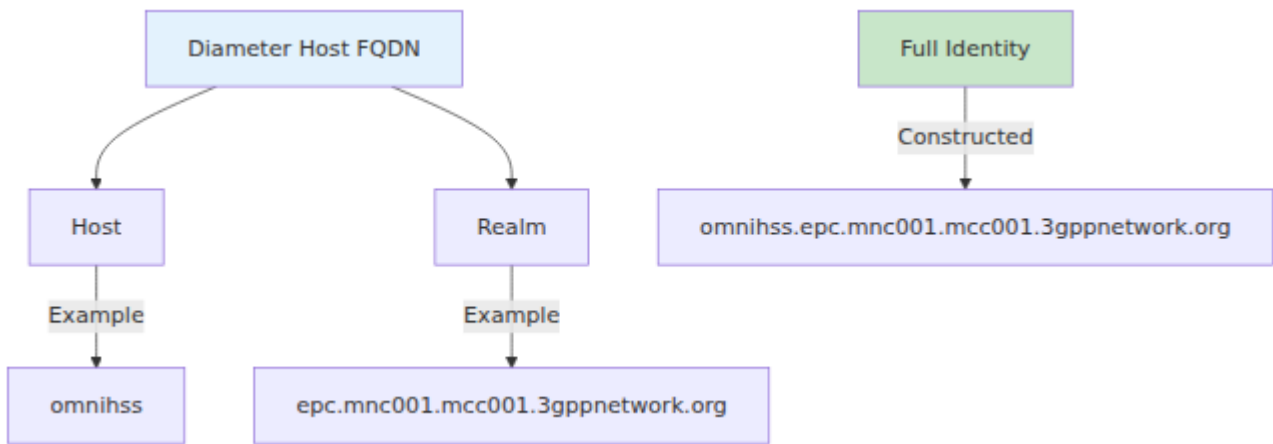
  # Stop Diameter service during database outages
  stop_diameter_on_database_failure: true,

  # License enforcement configuration
  license_enforced: true,
  license_module: LicenseClient
```

### HSS Core Parameters:

Parameter	Description	Default	Re
<code>ecto_repos</code>	List of Ecto repositories used by the application	<code>[Hss.Repo]</code>	Re da op
<code>send_clr_on_mme_change</code>	Send Cancel Location Request when subscriber changes MME	<code>true</code>	Ke pro
<code>stop_diameter_on_database_failure</code>	Disable Diameter service if database becomes unavailable	<code>true</code>	En col
<code>license_enforced</code>	Enable license enforcement	<code>true</code>	Re pro
<code>license_module</code>	Module handling license checks	<code>LicenseClient</code>	Do

# PLMN Code Format



## Examples:

- AT&T (USA): MCC=310, MNC=410
- Verizon (USA): MCC=311, MNC=480
- Vodafone (UK): MCC=234, MNC=15
- Test Network: MCC=001, MNC=01

# Network Interface Binding

```
# config/runtime.exs

# Diameter interface
listen_ip: System.get_env("DIAMETER_LISTEN_IP", "0.0.0.0"), # All
interfaces
# Or specific interface:
# listen_ip: "10.7.25.186",

# API interface
config :hss, HssWeb.Api.Endpoint,
  http: [
    ip: {0, 0, 0, 0}, # All interfaces
    port: 8443
  ]

# Control Panel interface
config :hss, HssWeb.ControlPanel.Endpoint,
  http: [
    ip: {0, 0, 0, 0}, # All interfaces
    port: 7443
  ]
```

## Interface Binding Options:

Binding Choice

0.0.0.0  
(All Interfaces)

Management IP  
(e.g., 192.168.1.10)

127.0.0.1  
(Localhost Only)

Accessible from  
any network

Accessible only from  
management network

Accessible only  
from server itself

---

# IMS Configuration

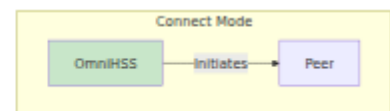
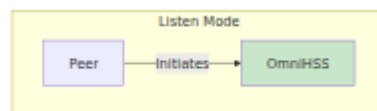
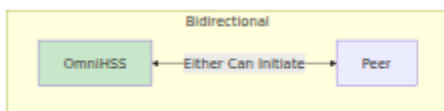
## S-CSCF Selection Configuration

```
# config/runtime.exs

config :hss, :ims, %{
  scscf: %{
    # Selection method: :random_peer or :round_robin
    selection_method: :random_peer,

    # List of available S-CSCF peers
    peers: [
      %{
        host:
        "sip:scscf01.ims.mnc001.mcc001.3gppnetwork.org:5060",
        capabilities: [] # Optional: capability matching
      },
      %{
        host:
        "sip:scscf02.ims.mnc001.mcc001.3gppnetwork.org:5060",
        capabilities: []
      }
    ]
  }
}
```

## S-CSCF Selection Methods



**Selection Methods:**

Method	Description	Use Case
<code>:random_peer</code>	Randomly selects an S-CSCF	Even load distribution
<code>:round_robin</code>	Sequentially assigns S-CSCFs	Predictable distribution

## IMS Realm Configuration

Typically, IMS uses a separate realm from EPC:

```
# EPC Realm
"epc.mnc001.mcc001.3gppnetwork.org"

# IMS Realm
"ims.mnc001.mcc001.3gppnetwork.org"
```

---

## EIR Configuration

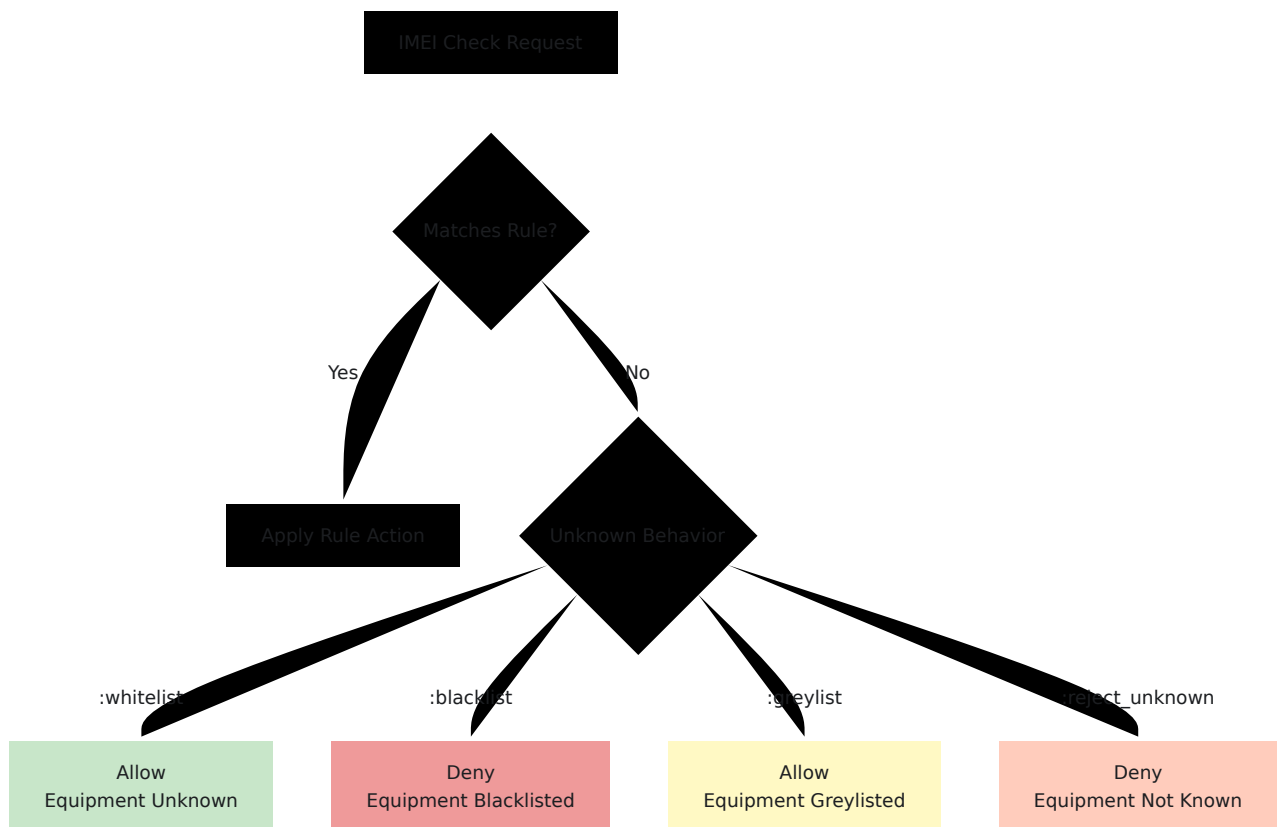
See [EIR Documentation](#) for complete equipment identity checking details.

## Equipment Identity Register Settings

```
# config/runtime.exs

config :hss, :eir, %{
  # Behavior for unknown equipment (no matching rule)
  unknown_equipment_behaviour: :whitelist
  # Options:
  #   :whitelist - Allow unknown equipment
  #   :blacklist - Block unknown equipment
  #   :greylist - Track but allow unknown equipment
  #   :reject_unknown_equipment - Reject with specific result code
}
```

# Unknown Equipment Behavior



## Behavior Options:

Option	Result	Use Case
<code>:whitelist</code>	Allow all unknown IMEI	Open network, testing
<code>:blacklist</code>	Block all unknown IMEI	Moderate security
<code>:greylist</code>	Allow but track unknown IMEI	Monitoring mode
<code>:reject_unknown_equipment</code>	Reject with specific code	High security

**Recommendation:** Start with `:whitelist` during testing, move to `:greylist` for production monitoring, then `:blacklist` for strict security.

---

# API and Control Panel Configuration

## API Endpoint Configuration

```
# config/config.exs

config :hss, HssWeb.Api.Endpoint,
  url: [host: "localhost"],
  render_errors: [view: HssWeb.ErrorView, accepts: ~w(json)],
  pubsub_server: Hss.PubSub,

# HTTPS configuration
https: [
  port: 8443,
  cipher_suite: :strong,
  certfile: "priv/cert/omnitouch.crt",
  keyfile: "priv/cert/omnitouch.pem"
]
```

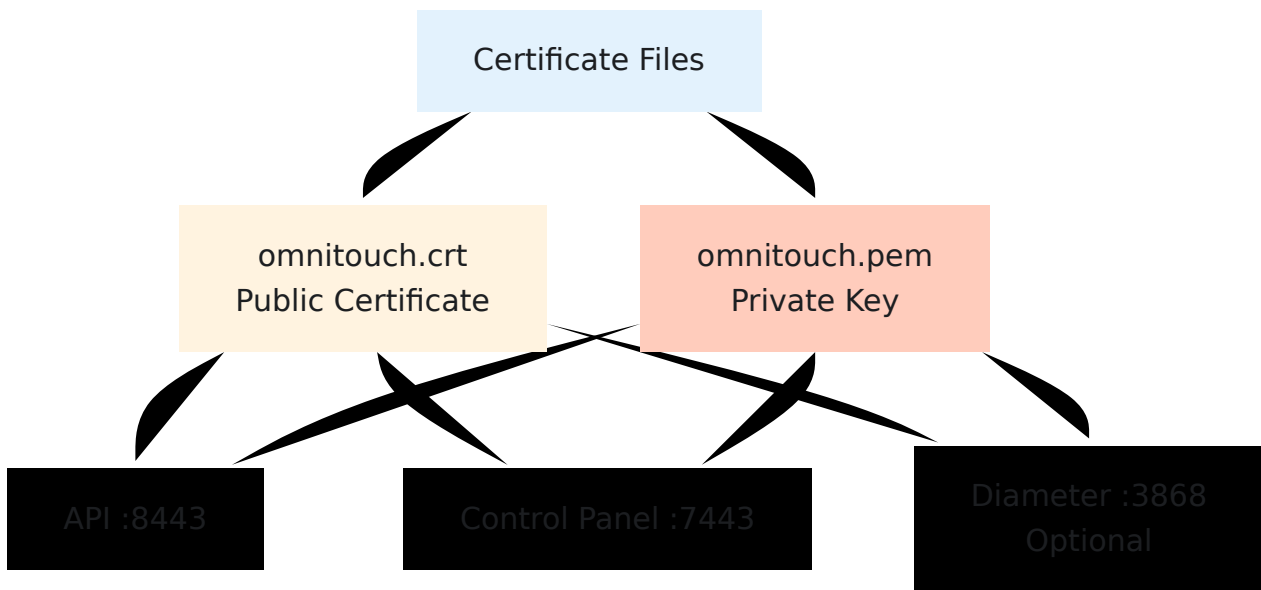
## Control Panel Configuration

```
# config/config.exs

config :hss, HssWeb.ControlPanel.Endpoint,
  url: [host: "localhost"],
  render_errors: [view: HssWeb.ErrorView, accepts: ~w(html json)],
  pubsub_server: Hss.PubSub,
  live_view: [signing_salt: "some-secret"],

# HTTPS configuration
https: [
  port: 7443,
  cipher_suite: :strong,
  certfile: "priv/cert/omnitouch.crt",
  keyfile: "priv/cert/omnitouch.pem"
]
```

# TLS Certificate Configuration



## Certificate Requirements:

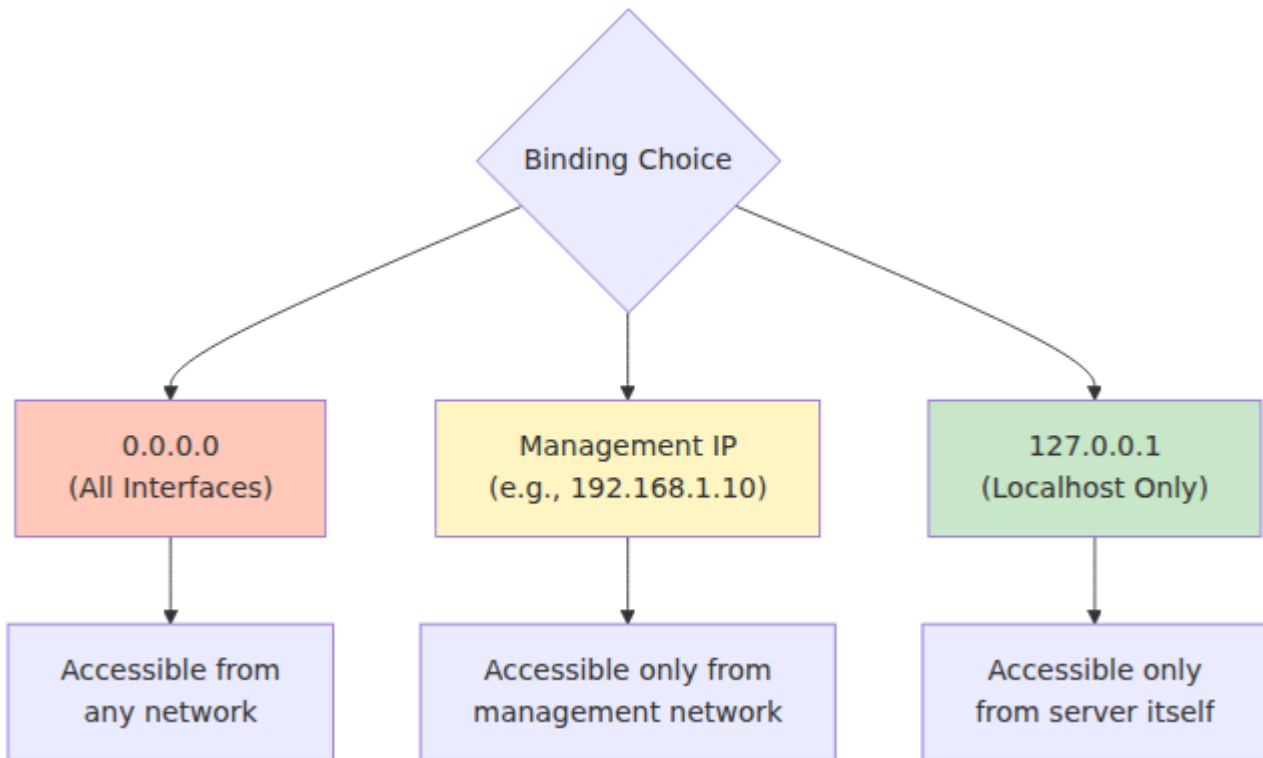
- Valid X.509 certificate
- Matching private key
- Include intermediate certificates if needed
- CN or SAN must match hostname

## For Production:

```
https: [  
  port: 8443,  
  cipher_suite: :strong,  
  certfile: System.get_env("TLS_CERT_FILE",  
    "/etc/ssl/certs/omnihss.crt"),  
  keyfile: System.get_env("TLS_KEY_FILE",  
    "/etc/ssl/private/omnihss.key"),  
  cacertfile: System.get_env("TLS_CA_FILE", "/etc/ssl/certs/ca-  
bundle.crt")  
]
```

# Configuration Workflow

## Initial Deployment Configuration



## Configuration Checklist

### Essential Configuration

- Database connection (hostname, credentials)
- Home PLMN (MCC, MNC)
- Diameter host and realm
- Diameter listen IP and port
- TLS certificates for API and Control Panel
- License client configuration (server URLs, licensee, product\_name)
- HSS core settings (send\_clr\_on\_mme\_change, stop\_diameter\_on\_database\_failure)

### Network Element Integration

- Diameter peers configured (if using connect mode)

- Firewall rules allow Diameter traffic (port 3868)
- Firewall rules allow HTTPS traffic (ports 7443, 8443)
- DNS resolution for Diameter identities

### **IMS Configuration (if using IMS features)**

- S-CSCF peer list configured
- S-CSCF selection method chosen
- IMS realm configured

### **Optional Configuration**

- EIR behavior configured
- Database pool size tuned
- Network interface binding restricted

## **Verifying Configuration**

After modifying configuration:

#### **1. Syntax Check:**

```
Check logs for configuration loading errors
```

#### **2. Control Panel Access:**

```
Access https://[hostname]:7443  
Verify Overview page loads
```

#### **3. API Access:**

```
curl -k https://[hostname]:8443/api/status
```

#### **4. Diameter Status:**

Check Control Panel Diameter page  
Verify peer connections

## 5. **Database Connectivity:**

Check Control Panel for subscriber data  
Or connect directly to SQL Database

---

# Complete Runtime Configuration

# Example

```
# config/runtime.exs - Complete production example

import Config

#
=====
# DATABASE CONFIGURATION
#
=====
config :hss, Hss.Repo,
  username: System.fetch_env!("DATABASE_USERNAME"),
  password: System.fetch_env!("DATABASE_PASSWORD"),
  hostname: System.get_env("DATABASE_HOSTNAME", "db.omnihss.internal"),
  database: System.get_env("DATABASE_NAME", "omnihss"),
  port: String.to_integer(System.get_env("DATABASE_PORT", "5432")),
  pool_size: String.to_integer(System.get_env("DATABASE_POOL_SIZE", "10")),
  timeout: 15_000,
  connect_timeout: 15_000,
  ssl: true,
  ssl_opts: [
    cacertfile: "/etc/ssl/certs/postgres-ca.pem",
    verify: :verify_peer
  ]

#
=====
# LICENSE CLIENT CONFIGURATION
#
=====
config :license_client,
  license_server_api_urls: [System.get_env("LICENSE_SERVER_URL",
"https://license.example.com:8443/api")],
  licensee: System.get_env("LICENSE_ORGANIZATION", "Your Organization"),
  product_name: "omnihss"

#
=====
# HOME PLMN AND HSS CORE CONFIGURATION
#
=====
```

```

config :hss,
  ecto_repos: [Hss.Repo],
  home_plmn: %{
    mcc: System.get_env("HOME_PLMN_MCC", "001"),
    mnc: System.get_env("HOME_PLMN_MNC", "001")
  },
  send_clr_on_mme_change: true,
  stop_diameter_on_database_failure: true,
  license_enforced: true,
  license_module: LicenseClient

#
=====
# DIAMETER CONFIGURATION
#
=====
diameter_config = %{
  service_name: :omnitouch_hss,
  listen_ip: System.get_env("DIAMETER_LISTEN_IP", "10.7.25.186"),
  listen_port: String.to_integer(System.get_env("DIAMETER_LISTEN_PORT",
"3868")),
  host: System.get_env("DIAMETER_HOST", "omnihss01"),
  realm: System.get_env("DIAMETER_REALM",
"epc.mnc001.mcc001.3gppnetwork.org"),
  product_name: "OmniHSS",
  vendor_id: 10415,
  supported_vendor_ids: [5535, 10415],
  request_timeout: 5000,
  peers: [
    %{
      host: "mme01.epc.mnc001.mcc001.3gppnetwork.org",
      realm: "epc.mnc001.mcc001.3gppnetwork.org",
      ip: "10.7.25.100",
      port: 3868,
      transport: :sctp,
      applications: [:s6a]
    }
  ]
}

config :hss, :diameter, diameter_config

#
=====

```

```

# IMS CONFIGURATION
#
=====
config :hss, :ims, %{
  scscf: %{
    selection_method: :random_peer,
    peers: [
      %{host: "sip:scscf01.ims.mnc001.mcc001.3gppnetwork.org:5060"},
      %{host: "sip:scscf02.ims.mnc001.mcc001.3gppnetwork.org:5060"}
    ]
  }
}

#
=====
# EIR CONFIGURATION
#
=====
config :hss, :eir, %{
  unknown_equipment_behaviour: :whitelist
}

#
=====
# API ENDPOINT CONFIGURATION
#
=====
config :hss, HssWeb.Api.Endpoint,
  http: [ip: {0, 0, 0, 0}, port: 8443],
  https: [
    port: 8443,
    cipher_suite: :strong,
    certfile: System.get_env("TLS_CERT_FILE", "/etc/ssl/certs/omnihss"),
    keyfile: System.get_env("TLS_KEY_FILE", "/etc/ssl/private/omnihss"),
  ],
  url: [host: System.get_env("API_HOST", "api.omnihss.internal"), port: 8443]

#
=====
# CONTROL PANEL ENDPOINT CONFIGURATION
#
=====
config :hss, HssWeb.ControlPanel.Endpoint,

```

```
http: [ip: {0, 0, 0, 0}, port: 7443],
https: [
  port: 7443,
  cipher_suite: :strong,
  certfile: System.get_env("TLS_CERT_FILE", "/etc/ssl/certs/omnihss"),
  keyfile: System.get_env("TLS_KEY_FILE", "/etc/ssl/private/omnihss"),
],
url: [host: System.get_env("CP_HOST", "hss.omnihss.internal"), port
```

---

[← Back to Operations Guide](#) | [Next: Entity Relationships](#) →

# OmniHSS Control Panel Guide

[← Back to Operations Guide](#)

---

## Table of Contents

- [Control Panel Overview](#)
  - [Accessing the Control Panel](#)
  - [Overview Page](#)
  - [Diameter Page](#)
  - [Application Page](#)
  - [Configuration Page](#)
  - [Navigation and Interface](#)
- 

## Control Panel Overview

The OmniHSS Control Panel is a web-based monitoring interface that provides real-time visibility into system status, subscriber activity, and Diameter connectivity. Built with Phoenix LiveView, it automatically updates without requiring page refreshes.

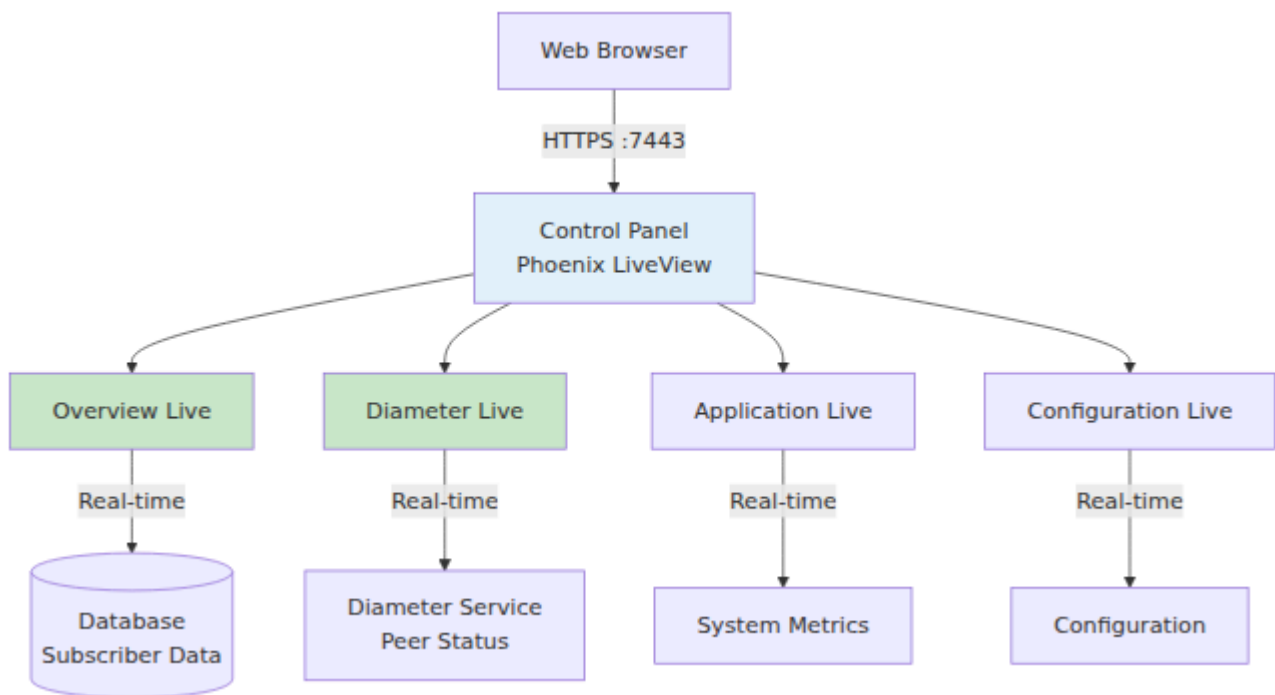
## Key Features

- **Real-time Updates** - Auto-refreshes every second
- **Subscriber Monitoring** - View active subscribers and their current state
- **Diameter Status** - Monitor peer connections in real-time
- **System Resources** - Track application performance
- **Configuration Viewer** - Inspect runtime configuration

## Access Information

URL: `https://[hostname]:7443`  
Protocol: HTTPS Only  
Port: 7443 (configurable)  
Certificate: Configured in `config/config.exs`

## Control Panel Architecture



## Accessing the Control Panel

### Initial Access

1. Open a web browser
2. Navigate to `https://[hostname]:7443`
3. Accept the TLS certificate (if self-signed)
4. You will be presented with the Overview page by default

# TLS Certificate Warnings

If using self-signed certificates, browsers will show security warnings. This is expected for internal deployments.

**For Production:** Use certificates signed by a trusted Certificate Authority.

## Network Requirements

- **Port 7443** must be accessible from your management network
- **HTTPS** is mandatory - HTTP is not supported
- **Firewall rules** must allow traffic to port 7443

## Browser Compatibility

The Control Panel uses modern web technologies (LiveView, WebSockets):

- Chrome/Chromium (recommended)
- Firefox
- Safari
- Edge

**Note:** Internet Explorer is not supported.

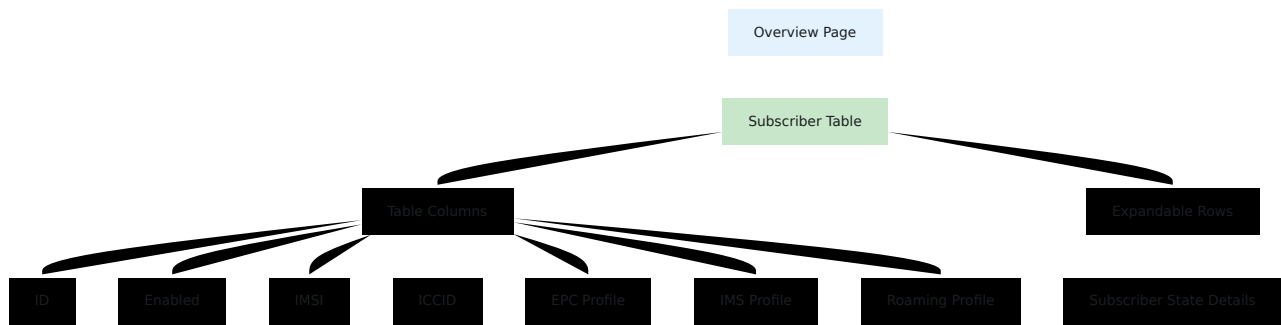
---

## Overview Page

**URL:** `https://[hostname]:7443/overview`

The Overview page displays all subscribers and their real-time state information.

# Page Layout



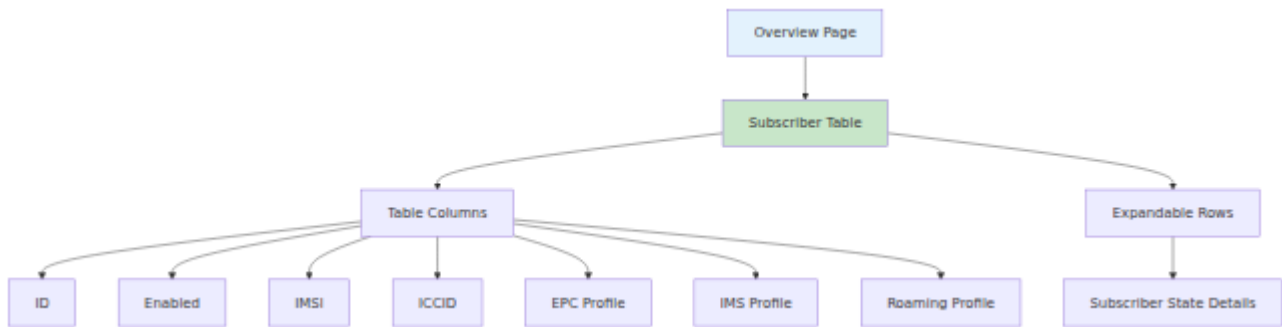
## Table Columns

Column	Description	Values
<b>ID</b>	Subscriber database ID	Integer
<b>Enabled</b>	Service status	✓ (enabled) / ✗ (disabled)
<b>IMSI</b>	International Mobile Subscriber Identity	14-15 digits
<b>ICCID</b>	SIM card ID	19-20 digits or "N/A"
<b>EPC Profile</b>	Data service profile name	Profile name or ID
<b>IMS Profile</b>	Voice service profile name	Profile name, ID, or "N/A"
<b>Roaming Profile</b>	Roaming policy name	Profile name, ID, or "N/A"

## Expandable Row Details

Click on any row to expand and view detailed subscriber state:

### Location Information



## Fields:

- **MCC** - Mobile Country Code (3 digits)
- **MNC** - Mobile Network Code (2-3 digits)
- **TAC** - Tracking Area Code
- **Cell ID** - Serving cell identifier
- **eNodeB ID** - Base station identifier
- **ECI** - E-UTRAN Cell Identifier

## Network Information

### Fields:

- **Last Seen MME** - Current serving MME hostname
- **Last Seen Realm** - Diameter realm of MME
- **RAT Type** - Radio Access Technology (e.g., "E-UTRAN" for LTE)
- **Last Seen At** - Timestamp of last Diameter message

## IMS Information

### Fields:

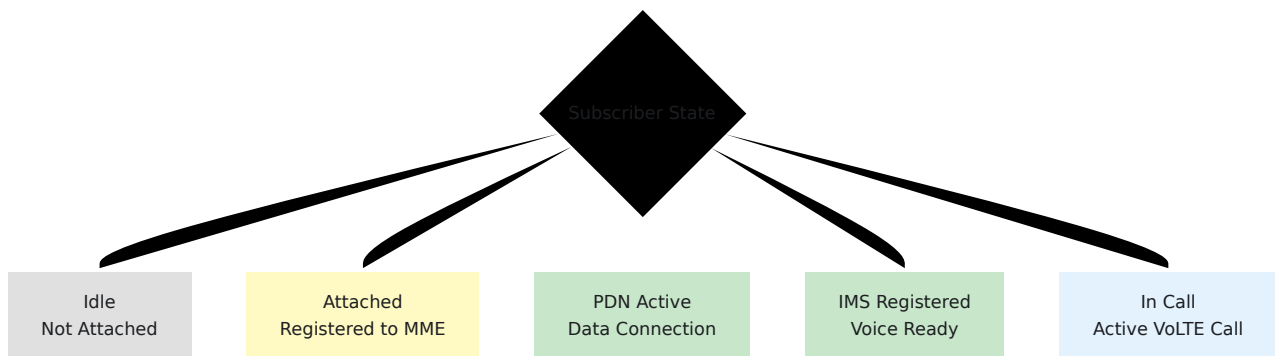
- **Assigned S-CSCF** - Currently assigned S-CSCF SIP URI
- **IMS Public Identity** - SIP URI (e.g., sip:+14155551234@ims.example.com)
- **Last Seen P-CSCF** - Last P-CSCF that contacted HSS
- **Last Seen I-CSCF** - Last I-CSCF that contacted HSS

## Session Information

### Fields:

- **PDN Sessions** - Number of active data connections
- **Active Calls** - Number of active VoLTE calls

## State Indicators



### How to identify state:

- **Idle:** No location information, no MME
- **Attached:** Last Seen MME present, location info available
- **PDN Active:** PDN sessions count > 0
- **IMS Registered:** Assigned S-CSCF present
- **In Call:** Active calls count > 0

## Auto-Refresh

The Overview page automatically refreshes **every 1 second** to show real-time updates.

### Visual indicators:

- New data appears without page reload
- Timestamps update in real-time
- No manual refresh needed

## Use Cases

### 1. Monitor Active Subscribers

- See which subscribers are currently attached

- Check current serving network (for roaming)
- Verify IMS registration status

## 2. Troubleshooting

- Verify subscriber is enabled
- Check last seen timestamp (is subscriber responsive?)
- Confirm profile assignments
- View current location information

## 3. Capacity Monitoring

- Count total attached subscribers
- Monitor PDN session counts
- Track active VoLTE calls

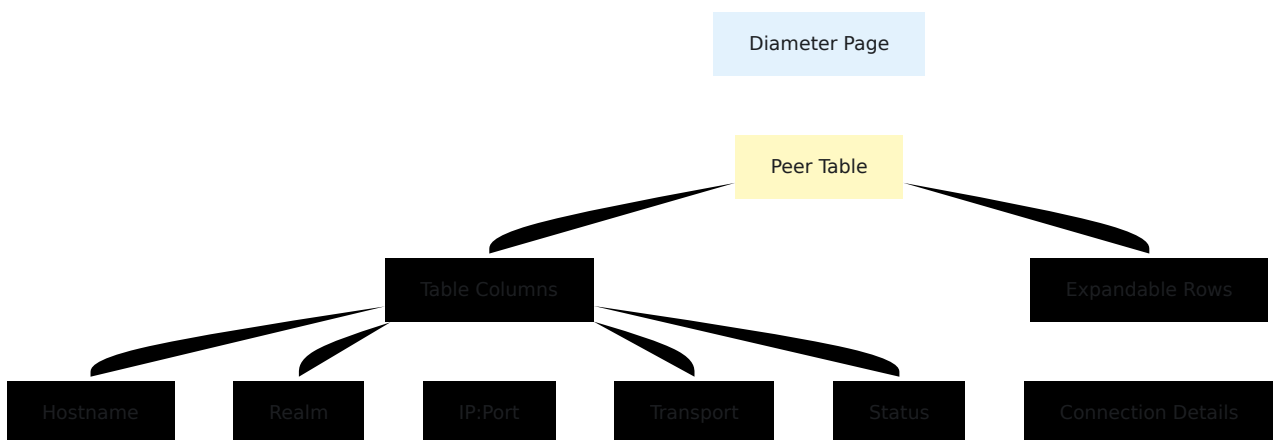
---

# Diameter Page

**URL:** `https://[hostname]:7443/diameter`

The Diameter page shows real-time status of all Diameter peer connections.

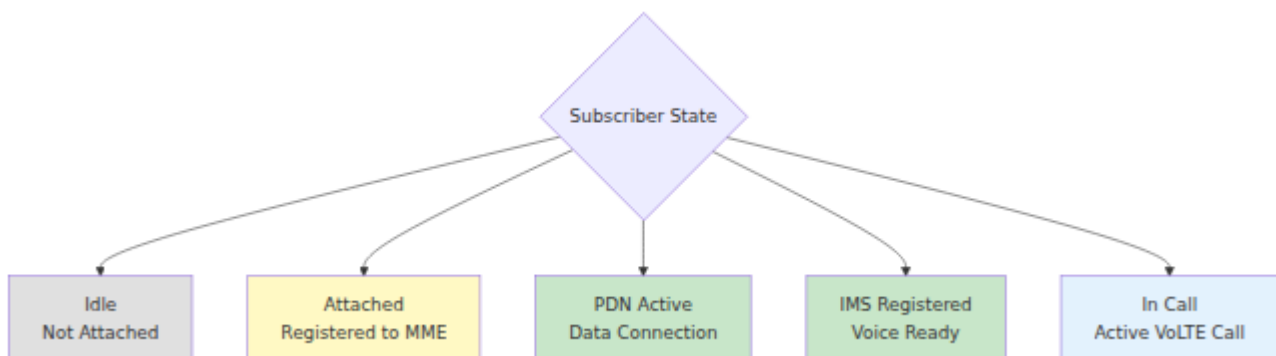
## Page Layout



## Table Columns

Column	Description	Values
<b>Hostname</b>	Diameter peer hostname	FQDN
<b>Realm</b>	Diameter realm	Domain name
<b>IP:Port</b>	Network address	IP address and port
<b>Transport</b>	Transport protocol	TCP or SCTP
<b>Status</b>	Connection status	Connected / Disconnected

## Connection Status



## Expandable Row Details

Click on any peer to view additional information:

### Connection Information:

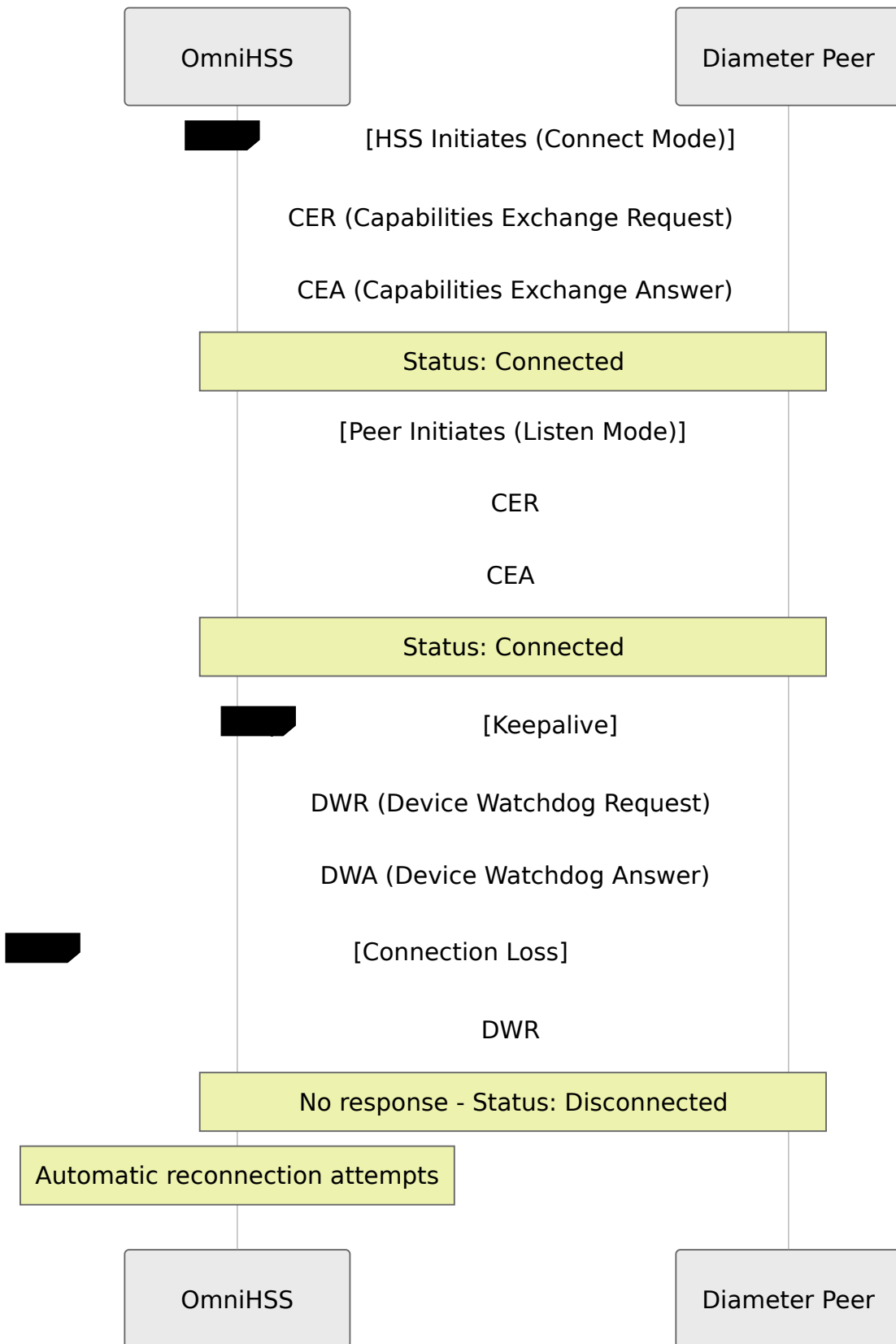
- **Connection Type** - Initiated by HSS or peer
- **Product Name** - Peer's product identification
- **Application IDs** - Supported Diameter applications

### Application ID Examples:

- 16777251 - S6a (MME)

- 16777238 - Gx (P-GW)
- 16777216 - Cx (I-CSCF, S-CSCF)
- 16777217 - Sh (Application Server)
- 16777236 - Rx (P-CSCF)
- 16777252 - S13 (EIR client, if external)

# Peer Connection Flow



# Auto-Refresh

The Diameter page automatically refreshes **every 1 second**.

## Use Cases

### 1. Verify Connectivity

- Ensure all expected peers are connected
- Identify disconnected peers immediately
- Monitor for flapping connections

### 2. Troubleshooting

- Check if peer is reachable
- Verify transport protocol (TCP vs SCTP)
- Confirm application IDs match expectations
- Identify which side initiated connection

### 3. Capacity Planning

- Count total connected peers
- Monitor for connection stability
- Plan for additional peer capacity

## Common Issues

### Peer Shows Disconnected

#### Possible Causes:

1. Network connectivity issue
2. Peer is down or restarting
3. Firewall blocking traffic
4. Diameter configuration mismatch
5. Certificate issue (if using TLS)

#### Troubleshooting Steps:

1. Check network connectivity: `ping [peer-ip]`
2. Verify port is reachable: `telnet [peer-ip] 3868`
3. Check firewall rules
4. Review HSS logs for error messages
5. Verify peer's Diameter configuration matches HSS

## Peer Connects and Disconnects Repeatedly

### Possible Causes:

1. Network instability
2. Keepalive timeout mismatch
3. Peer resource issues
4. Diameter application mismatch

### Troubleshooting Steps:

1. Check network stability
  2. Review keepalive timers on both sides
  3. Check peer system resources
  4. Verify application IDs match on both sides
- 

# Application Page

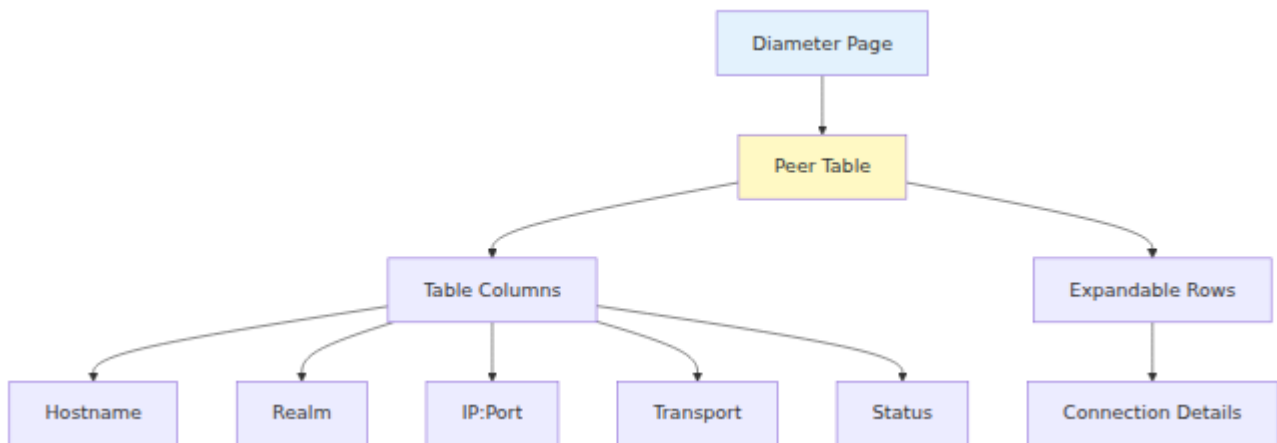
**URL:** `https://[hostname]:7443/application`

The Application page provides system-level monitoring and resource usage information.

## Features

- **Process Information** - Erlang VM process count and memory
- **System Memory** - Total and used memory
- **Application Uptime** - How long OmniHSS has been running
- **Erlang VM Version** - Runtime version information

# Key Metrics



## Use Cases

### 1. Health Monitoring

- Verify application is running
- Check for memory leaks (increasing memory over time)
- Monitor process count growth

### 2. Capacity Planning

- Track memory usage trends
- Plan for scale-out based on process count
- Verify adequate system resources

### 3. Troubleshooting

- Identify resource exhaustion
- Check if restart is needed
- Verify Erlang VM version

---

# Configuration Page

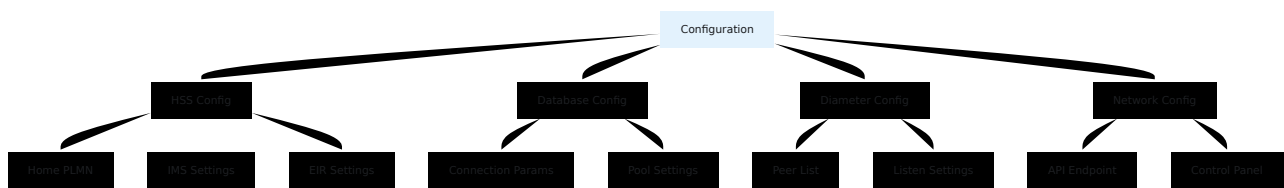
**URL:** `https://[hostname]:7443/configuration`

The Configuration page displays the current runtime configuration of OmniHSS.

## Features

- **View Configuration** - Inspect all configuration parameters
- **Search Configuration** - Find specific settings
- **Environment Variables** - See resolved values

## Configuration Categories



## Use Cases

### 1. Configuration Verification

- Verify runtime.exs settings are applied
- Confirm database connection parameters
- Check Diameter peer configuration

### 2. Troubleshooting

- Identify misconfiguration
- Verify environment variables are set correctly
- Compare expected vs actual configuration

### 3. Documentation

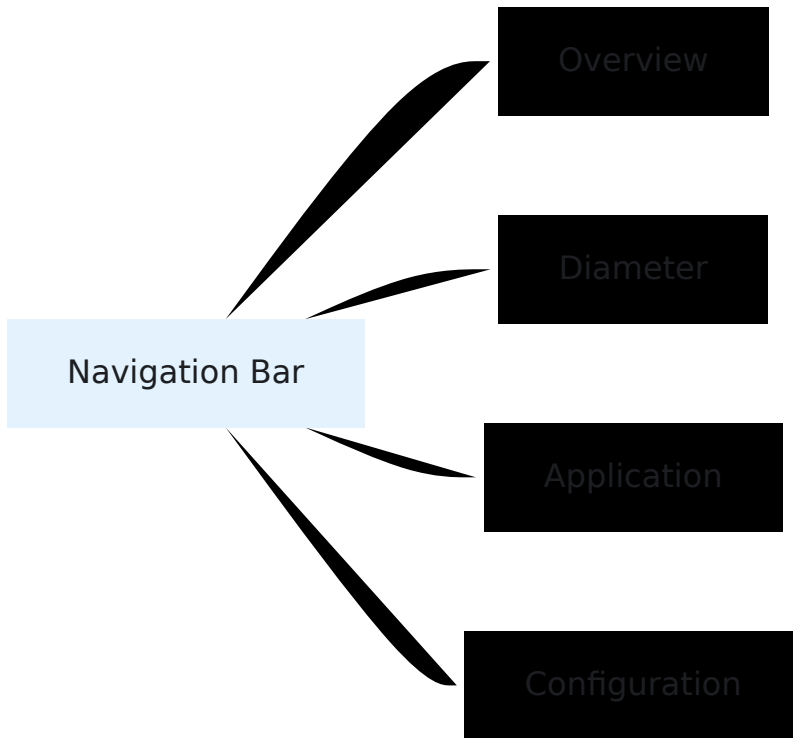
- Export current configuration for documentation
- Share configuration with support team

**Security Note:** Configuration page may display sensitive information (database passwords, keys). Restrict access appropriately.

---

# Navigation and Interface

## Top Navigation Bar



**Navigation is always visible** at the top of the page for quick access.

## Keyboard Shortcuts

While the Control Panel doesn't implement custom keyboard shortcuts, standard browser shortcuts work:

- **Ctrl+R / F5** - Manual page refresh (though auto-refresh makes this unnecessary)
- **Ctrl+F** - Search on page
- **Ctrl+T** - Open new tab (for multiple pages)

## Multi-Tab Monitoring

You can open multiple Control Panel pages in separate browser tabs for simultaneous monitoring:

**Example Setup:**

- Tab 1: Overview page (monitor subscribers)
- Tab 2: Diameter page (monitor connectivity)
- Tab 3: Application page (monitor resources)

All tabs will auto-update independently.

## Responsive Design

The Control Panel is optimized for desktop browsers. Mobile browsers are supported but may require horizontal scrolling for tables.

**Recommended Resolution:** 1920x1080 or higher for comfortable viewing.

---

# Monitoring Best Practices

## Daily Operations

### 1. Start of Shift

- Open Control Panel Overview page
- Verify expected number of subscribers are attached
- Check Diameter page - all peers connected

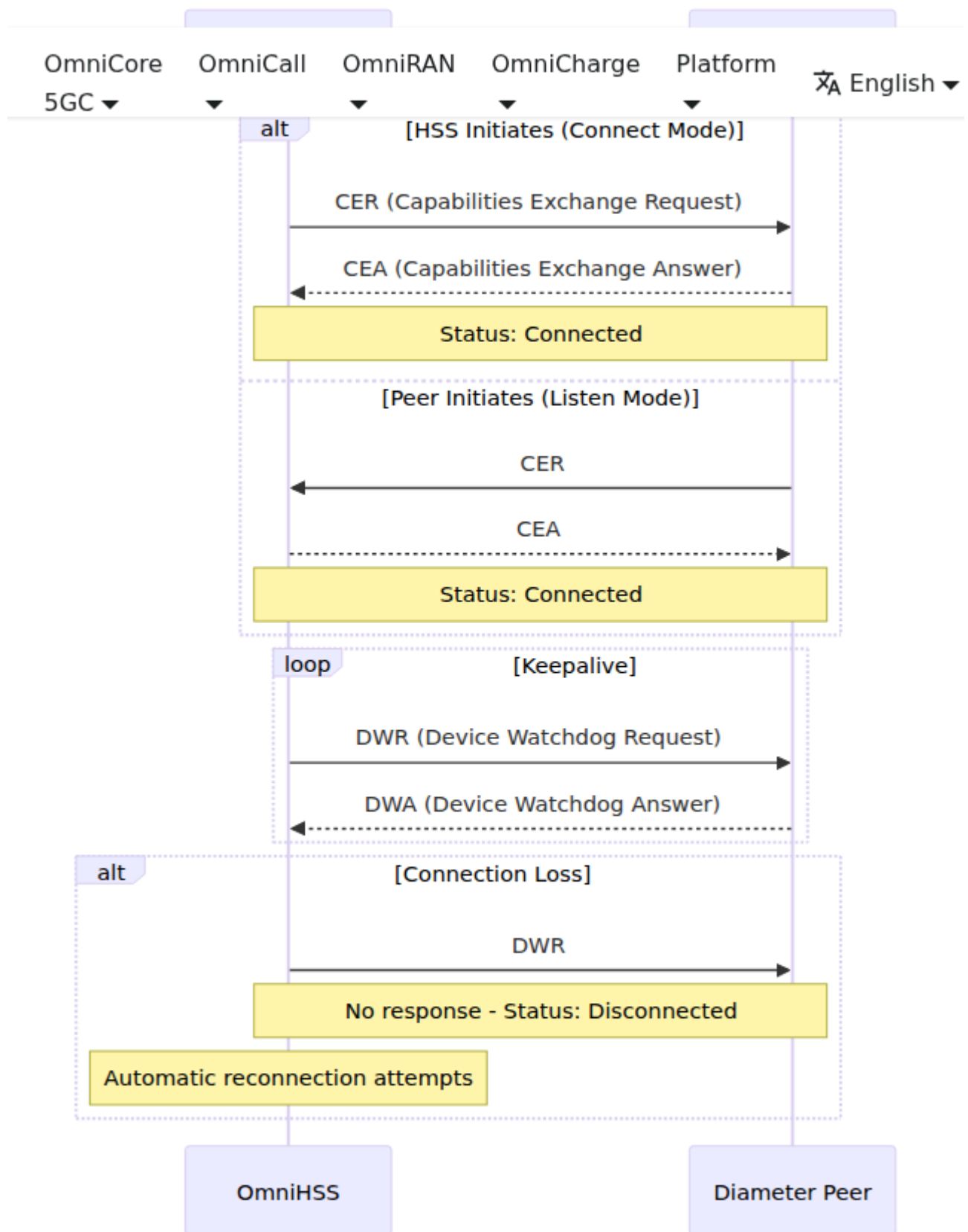
### 2. During Shift

- Keep Overview page open for real-time monitoring
- Watch for unusual state changes
- Monitor for disconnected peers on Diameter page

### 3. End of Shift

- Verify system is stable
- Check Application page for resource usage trends
- Document any anomalies

# Troubleshooting Workflow



# Alert Thresholds

Establish monitoring thresholds for proactive alerting:

Metric	Warning	Critical
Disconnected Diameter Peers	1 peer	2+ peers or critical peer
Memory Usage	> 80%	> 90%
Subscriber Authentication Failures	> 5%	> 10%
Process Count	> 80% of limit	> 95% of limit

---

[← Back to Operations Guide](#) | [Next: Metrics & Monitoring](#) →

# Database Replication

[← Back to Operations Guide](#)

---

## Table of Contents

- [Overview](#)
  - [How It Works](#)
  - [Deployment Architectures](#)
  - [Configuration Reference](#)
  - [Peer Management](#)
  - [Monitoring](#)
  - [Troubleshooting](#)
- 

## Overview

OmniHSS uses **PostgreSQL** with **bidirectional logical replication** for multi-site high-availability deployments. Each site operates as an independent read/write primary — there is no single master or read-only replicas.

## Key Characteristics

- **Both Sites Read/Write:** Every site has a full primary database. Local writes are instant regardless of inter-site link quality
- **Asynchronous Replication:** Changes replicate via PostgreSQL's Write-Ahead Log (WAL). No write is blocked waiting for a remote site to acknowledge
- **Partition Tolerant:** If the inter-site link goes down, both sites continue operating independently. WAL segments queue locally and replay automatically when the link recovers

- **No Quorum Required:** Unlike synchronous cluster approaches, there is no concept of quorum. A single-site deployment operates identically to a multi-site deployment
- **UUID Primary Keys:** All records use UUIDv4 primary keys, eliminating ID collisions between sites without coordination

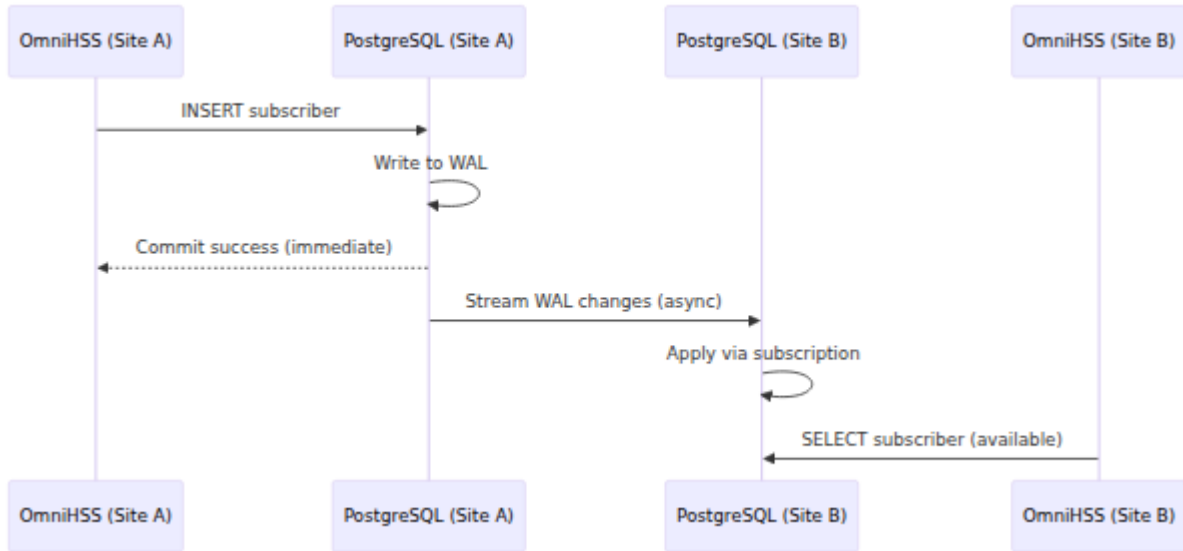
## How This Differs from Synchronous Replication

	<b>Synchronous</b>	<b>Asynchronous Logical Replication</b>
Write latency	Blocked by slowest node	Local only — sub-millisecond
Link failure	Minority site goes read-only	Both sites continue read/write
Quorum needed	Yes (majority of nodes)	No
Data consistency	Strong (all nodes identical)	Eventual (brief replication lag)
Suitable for satellite links	No (latency kills write performance)	Yes (WAL queues and catches up)

---

# How It Works

## Logical Replication Flow



## Publication / Subscription Model

Each site has:

- A **publication** that broadcasts all table changes (except `schema_migrations`)
- A **subscription** to every peer site's publication



## WAL Queuing During Link Outage

When the inter-site link is unavailable:

1. Local writes continue without interruption
2. WAL segments accumulate on the publishing side
3. The replication slot tracks where each subscriber left off
4. When the link recovers, queued WAL replays from the last acknowledged position
5. Both sites converge to identical state

WAL retention is bounded by `max_slot_wal_keep_size` to prevent unbounded disk growth during extended outages.

## Conflict Prevention

UUID primary keys eliminate the primary source of conflicts in bidirectional replication. Since each site generates globally unique identifiers independently, INSERT operations never collide.

The `origin = none` subscription option prevents replication loops — changes that arrived via replication are not re-published to other subscribers.

The `schema_migrations` table is excluded from publications because both sites run identical migrations independently.

## Last-Write-Wins for Dynamic State

Dynamic state tables (`subscriber_state`, `pdn_session`, `lte_call`) are updated frequently by Diameter signalling — every ULR updates the serving MME, every CCR updates the PGW session, every AAR updates the active call.

In a multi-site deployment, a subscriber is served by one site at a time. When the subscriber moves between sites, both sites may have updates to the same `subscriber_state` row from different times. Without conflict resolution, the replication order would determine which value survives — potentially overwriting the current serving MME with a stale value.

OmniHSS uses **Last-Write-Wins (LWW) triggers** on dynamic state tables:

```
CREATE TRIGGER lww_subscriber_state_trigger
  BEFORE UPDATE ON subscriber_state
  FOR EACH ROW
  EXECUTE FUNCTION lww_subscriber_state();
```

The trigger compares the `updated_at` timestamp of the incoming replicated update against the existing row. If the existing row is newer, the update is silently suppressed. This ensures:

- When a subscriber moves from Site A to Site B, Site B's newer update always wins
- Stale updates from Site A (from before the subscriber moved) are discarded
- Both sites converge to the same value — the most recent write

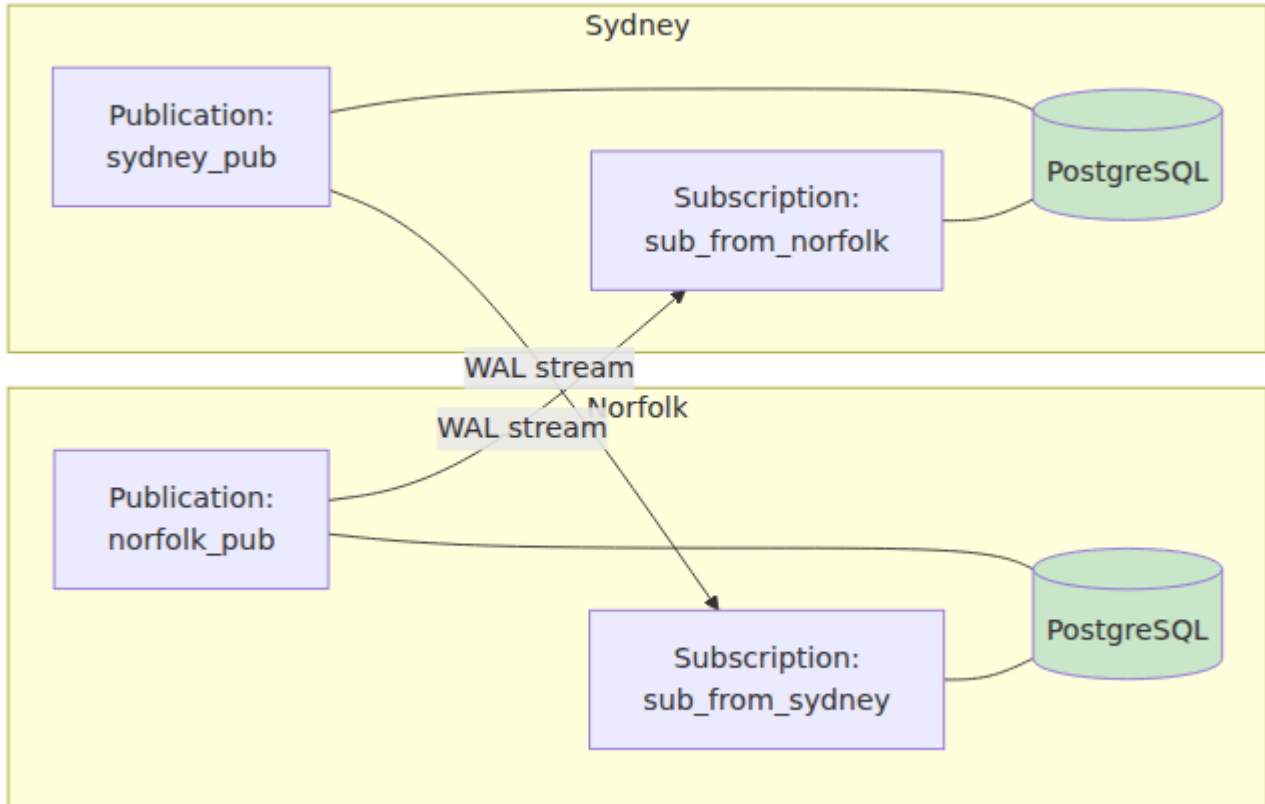
**Tables with LWW triggers:**

Table	Updated By	Frequency
subscriber_state	ULR, SAR, AIR, PUR	Every attach/location update
pdn_session	CCR-I, CCR-T	Every PDN session create/delete
lte_call	AAR, STR	Every VoLTE call setup/teardown

Static provisioning data (subscribers, profiles, keys, APNs) does not need LWW — these are only updated by the provisioning API, typically from a single management plane.

## Deployment Architectures

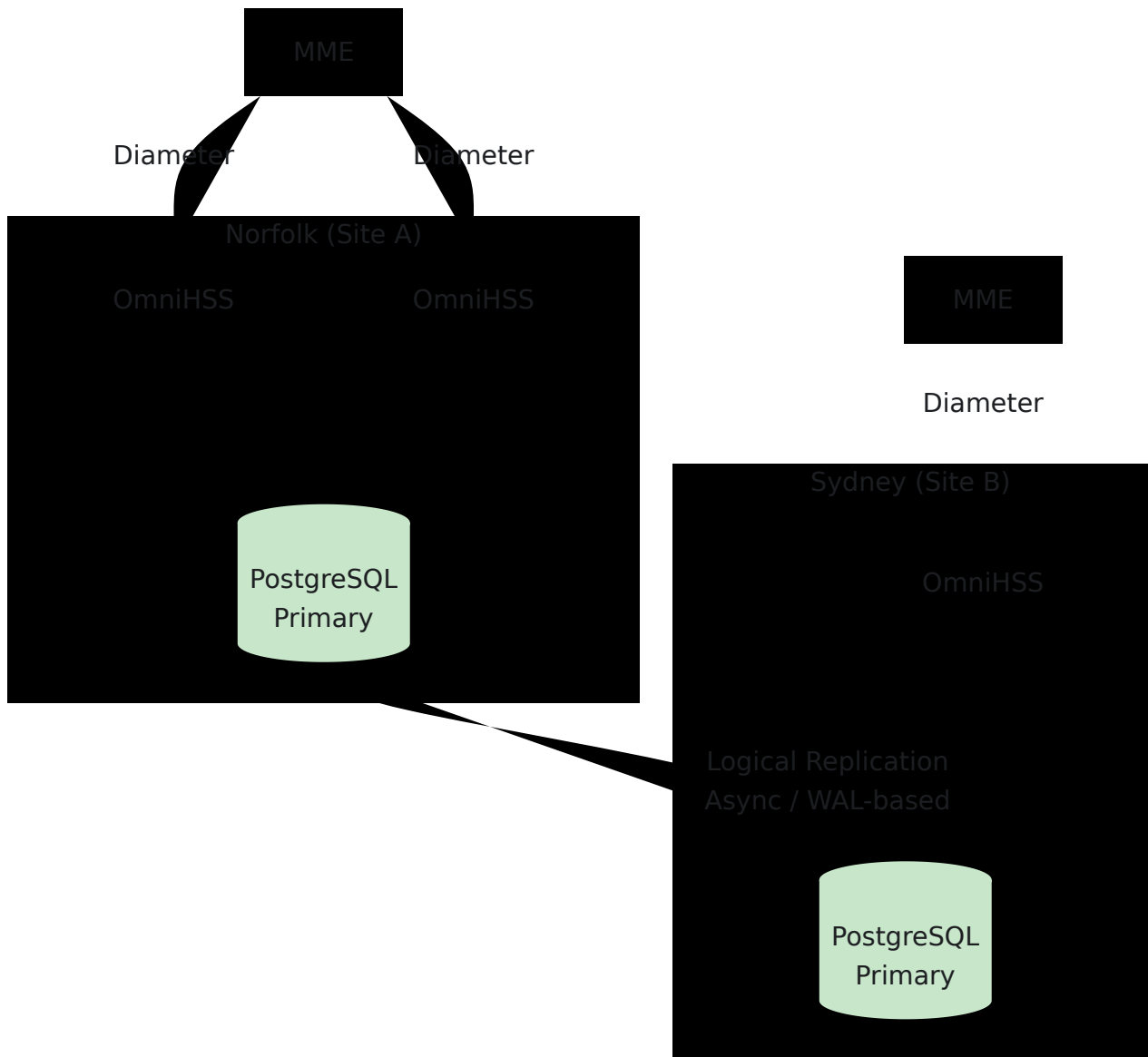
### Single Site (No Replication)



Multiple HSS application instances share a single PostgreSQL database. Standard PostgreSQL streaming replication can be used for local database

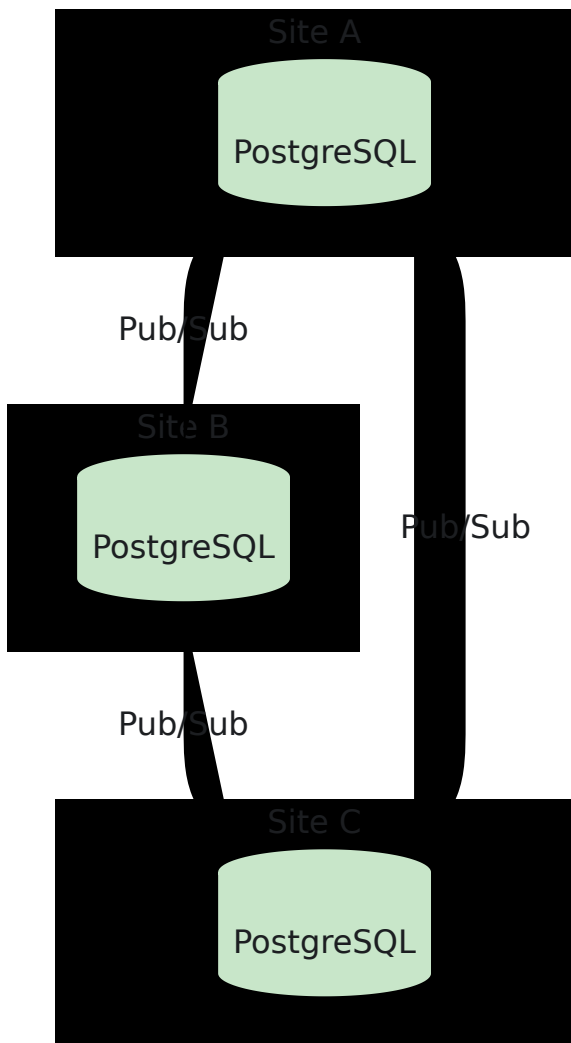
failover if required.

## Two-Site Deployment



Both sites are full primaries. Subscriber provisioning on either site replicates to the other. Each MME connects to its local HSS — there is no cross-site Diameter dependency.

# Multi-Site Deployment



Each site subscribes to every other site's publication. Peers are auto-discovered from Ansible inventory — adding a site requires only adding it to the `hss` inventory group and running the playbook.

## Network Requirements

Port	Protocol	Purpose
5432	TCP	PostgreSQL client and replication connections

---

# Configuration Reference

## Ansible Variables

Replication is configured per HSS group in your inventory:

```
hss:
  hosts:
    site-a-hss01:
      ansible_host: 10.80.12.140
    site-a-hss02:
      ansible_host: 10.80.12.141
  vars:
    omnihss:
      database_type: postgres
      database_host: localhost
      database_port: 5432
      database_name: omnihss
      database_username: omnihss
      database_password: "secure_password"
    replication:
      enabled: true
```

# Ansible Parameters

Parameter	Type	Required	Default	Description
<code>database_type</code>	String	No	<code>postgres</code>	Database backend. Must be <code>postgres</code> for replication.
<code>database_host</code>	String	No	<code>localhost</code>	PostgreSQL host. Use <code>localhost</code> when Postgres runs on the same host as OmniHSS.
<code>database_port</code>	Integer	No	<code>5432</code>	PostgreSQL port.
<code>database_name</code>	String	No	<code>omnihss</code>	Database name.
<code>database_username</code>	String	Yes	-	PostgreSQL user. Must have <code>CREATEDB</code> and <code>REPLICATION</code> privileges.
<code>database_password</code>	String	Yes	-	PostgreSQL password.

Parameter	Type	Required	Default	Description
<code>replication.enabled</code>	Boolean	No	<code>false</code>	Enable bidirectional logical replication. When <code>true</code> , the Ansible role configures publications, subscriptions, and <code>pg_hba.conf</code> entries automatically.

## Peer Discovery

Peers are discovered automatically from two sources:

1. **Local inventory group:** All other hosts in the `hss` Ansible group
2. **Remote sites:** Entries in `connected_sites.hss` from `prod_master_peers.yml`

No per-host peer configuration is required. Adding a host to the `hss` group or to `connected_sites.hss` and running the playbook is sufficient.

## Remote Site Peers (prod\_master\_peers.yml)

For inter-site replication across geographic locations:

```
# group_vars/prod_master_peers.yml
connected_sites:
  hss:
    - remote-site-hss01: 10.80.20.140
    - remote-site-hss02: 10.80.20.141
```

## PostgreSQL Configuration

The Ansible role configures the following PostgreSQL parameters automatically when replication is enabled:

Parameter	Value	Purpose
<code>wal_level</code>	<code>logical</code>	Required for logical replication. Includes full row data in WAL.
<code>max_wal_senders</code>	<code>10</code>	Maximum concurrent WAL sender processes (one per subscriber).
<code>max_replication_slots</code>	<code>10</code>	Maximum replication slots. Each peer uses one slot.
<code>listen_addresses</code>	<code>*</code>	Accept connections from peer sites.

The role also adds `pg_hba.conf` entries to allow replication connections from each discovered peer IP.

---

## Peer Management

### Adding a New Site

1. Add the new host(s) to the `hss` inventory group, or add to `connected_sites.hss` in `prod_master_peers.yml`
2. Run the OmniHSS playbook

The role will:

- Install PostgreSQL and OmniHSS on the new host
- Run database migrations
- Create a publication for the new site
- Create subscriptions to all existing peers
- Update `pg_hba.conf` on all existing peers to allow the new site
- Create subscriptions from existing peers to the new site

## Removing a Site

```
ansible-playbook -i hosts site.yml --tags hss-remove-peer \  
-e "peer_name=sydney confirm_remove=yes"
```

The `confirm_remove=yes` parameter is required as a safety measure. Without it, the playbook refuses to proceed and displays what would happen.

Removing a peer:

- Drops the subscription (stops incoming replication from that peer)
- Removes the peer's IP from `pg_hba.conf`
- Data already replicated from the peer remains in the local database
- The peer site's subscription to this site must be removed separately on the peer

## Adding a Peer Ad-Hoc

For cases where you need to add a peer outside the standard playbook flow:

```
ansible-playbook -i hosts site.yml --tags hss-add-peer \  
-e "peer_name=sydney peer_host=10.80.20.140"
```

---

# Monitoring

## Prometheus Metrics

The **postgres\_exporter** service runs on each HSS host on port `9187`, exposing PostgreSQL metrics to Prometheus.

### Replication Metrics

Metric	Type	Description
<code>pg_replication_slots_active</code>	Gauge	<code>1</code> if the replication slot is active (peer connected), <code>0</code> if inactive
<code>pg_replication_slots_pg_wal_lsn_diff</code>	Gauge	Replication lag in bytes. Distance between current WAL position and the peer's last confirmed position
<code>pg_stat_replication_pg_current_wal_lsn_bytes</code>	Gauge	Current WAL position on this node

### Example Prometheus Queries

```
# Replication slot active (1 = connected, 0 = disconnected)
pg_replication_slots_active{slot_name=~"sub_from_.*"}

# Replication lag in megabytes
pg_replication_slots_pg_wal_lsn_diff / 1048576

# Alert: replication slot inactive for more than 60 seconds
pg_replication_slots_active == 0
```

## Recommended Alerts

Alert	Condition	Severity	Description
Replication Slot Inactive	<code>pg_replication_slots_active == 0</code> for 60s	Critical	Peer is disconnected. WAL is accumulating.
WAL Lag Excessive	<code>pg_replication_slots_pg_wal_lsn_diff &gt; 104857600</code> for 5m	Warning	Replication lag exceeds 100MB. This may be due to network congestion or link degradation.
WAL Slot Lost	WAL status is <code>lost</code>	Critical	Slot has become invalid. Peer requires a full recovery.

## Ansible Validation

The OmniHSS playbook validates replication health at the end of every run. It asserts:

1. All subscription workers are alive
2. All replication slots are active

3. WAL lag is within threshold (100MB default)

4. WAL slot status is `reserved` or `extended` (not `lost`)

If any assertion fails, the playbook fails with a clear error message identifying the issue.

## Manual Validation

```
# Check subscription workers are alive
sudo -u postgres psql -d omnihss -c \
  "SELECT subname, pid IS NOT NULL as worker_alive FROM
pg_stat_subscription;"

# Check replication slots and lag
sudo -u postgres psql -d omnihss -c \
  "SELECT slot_name, active,
  pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn) as
lag_bytes,
  wal_status
  FROM pg_replication_slots WHERE slot_type = 'logical';"

# Compare subscriber counts between sites
sudo -u postgres psql -d omnihss -c "SELECT count(*) FROM
subscriber;"
```

---

## Troubleshooting

### Subscription Worker Dead

**Symptoms:** `pg_stat_subscription` shows `pid IS NULL` for a subscription. Ansible validation fails with "Dead workers."

**Common Causes:**

- Table exists on publisher but not subscriber (DDL mismatch)
- Duplicate key conflict from a manually inserted record
- Peer unreachable for extended period causing WAL slot invalidation

## Resolution:

1. Check PostgreSQL logs for the specific error:

```
journalctl -u postgresql | grep "logical replication" | tail -20
```

2. If DDL mismatch: apply the missing migration on this site, then the worker will auto-restart
3. If duplicate key: resolve the conflict manually, then re-enable the subscription:

```
ALTER SUBSCRIPTION sub_from_<peer> ENABLE;
```

## Replication Slot Inactive

**Symptoms:** `pg_replication_slots` shows `active = false`. Prometheus alert fires.

### Common Causes:

- Network connectivity issue between sites
- PostgreSQL on the peer site is stopped
- Firewall blocking port 5432 from the peer's IP

### Resolution:

1. Verify network connectivity to the peer:

```
pg_isready -h <peer_ip> -U <db_user> -d omnihss
```

2. Check the peer site's PostgreSQL is running
3. Verify firewall rules allow port 5432 from this site's IP

The subscription worker will automatically reconnect when the peer becomes reachable.

# WAL Slot Lost

**Symptoms:** `pg_replication_slots` shows `wal_status = 'lost'`. Ansible validation fails with "WAL SLOT CRITICAL."

**Cause:** The peer was disconnected for so long that the retained WAL exceeded `max_slot_wal_keep_size` and was cleaned up.

**Resolution:** The peer requires a full re-seed:

1. Drop the broken subscription on this site:

```
DROP SUBSCRIPTION sub_from_<peer>;
```

2. Drop the broken slot on the peer site:

```
SELECT pg_drop_replication_slot('sub_from_<this_site>');
```

3. Re-run the OmniHSS playbook — it will recreate the subscription with `copy_data = true` to perform an initial sync.

# Excessive WAL Lag

**Symptoms:** `pg_replication_slots_pg_wal_lsn_diff` is large and growing. Prometheus alert fires.

**Common Causes:**

- High-latency or congested inter-site link
- Peer site under heavy load (apply worker can't keep up)
- Large bulk operation (e.g. mass provisioning) generating significant WAL volume

**Resolution:**

1. Check if the lag is growing or stable:

```
SELECT slot_name,  
       pg_wal_lsn_diff(pg_current_wal_lsn(), confirmed_flush_lsn) as  
lag_bytes  
FROM pg_replication_slots WHERE slot_type = 'logical';
```

2. If stable: the peer is applying changes but behind. It will catch up.
3. If growing: investigate the network link or peer site load.
4. For bulk operations: lag during the operation is expected and will resolve after the operation completes.

## Schema Migration Coordination

**Important:** Database schema changes (migrations) do not replicate via logical replication. When deploying a new OmniHSS version that includes schema changes:

1. Apply the migration on **all sites** before deploying application code that depends on new columns/tables
2. The Ansible playbook handles this automatically — it runs migrations on every host
3. If deploying manually, run migrations on all sites first:

```
/opt/omnihss/bin/hss rpc "Hss.Command.Database.migrate()"
```

---

# OmniHSS Entity Relationships

[← Back to Operations Guide](#)

---

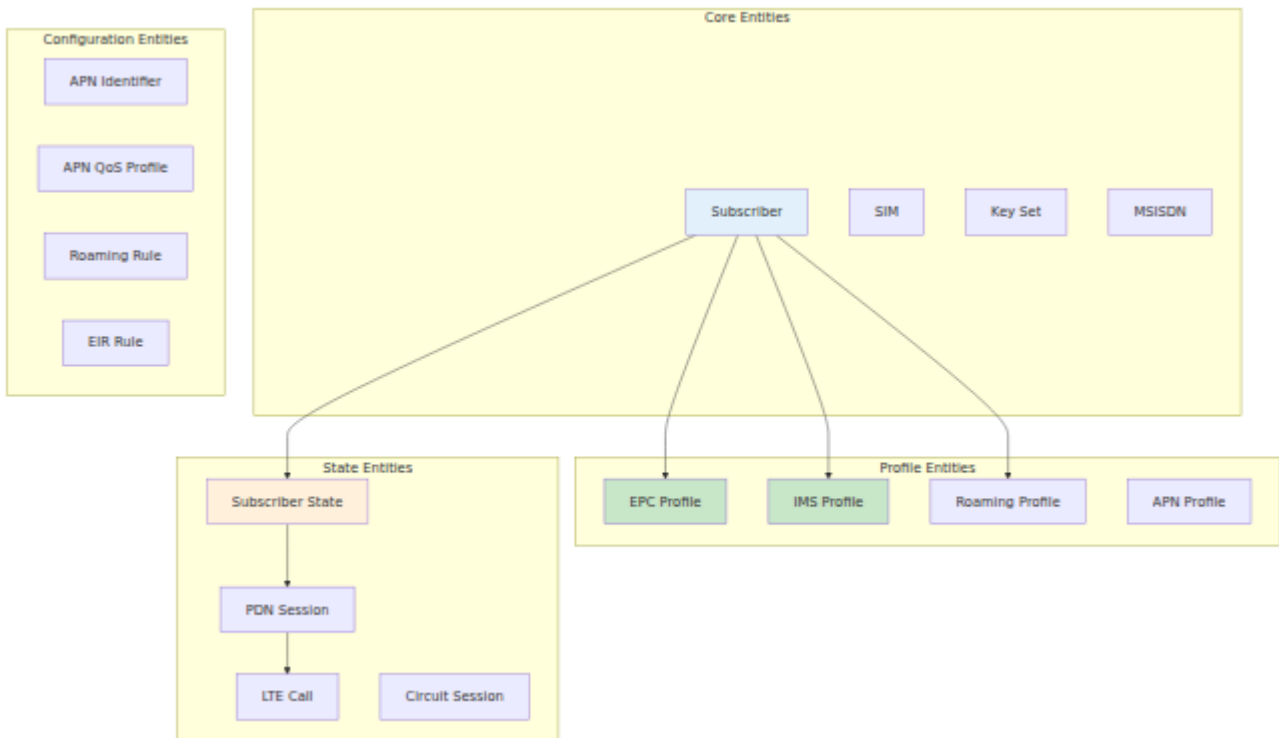
## Table of Contents

- [Entity Overview](#)
  - [Core Entities](#)
  - [Profile Entities](#)
  - [State Entities](#)
  - [Entity Relationship Diagrams](#)
  - [Entity Lifecycle](#)
  - [Data Flow Patterns](#)
- 

## Entity Overview

OmniHSS organizes subscriber data into logical entities with clear relationships. Understanding these entities is crucial for operational tasks like provisioning, troubleshooting, and capacity planning.

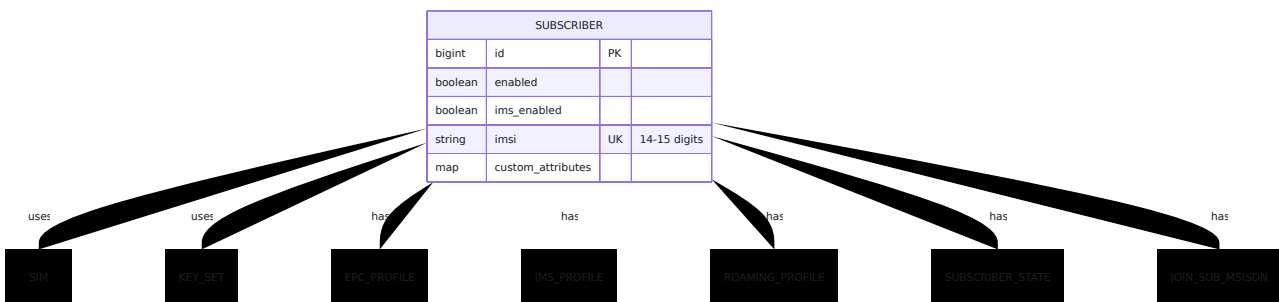
# Entity Categories



# Core Entities

## Subscriber

The **Subscriber** is the central entity representing a mobile user.



**Fields:**

Field	Type	Description	Constraints
<code>id</code>	bigint	Primary key	Auto-increment
<code>enabled</code>	boolean	Service enabled flag	Default: true
<code>ims_enabled</code>	boolean	IMS services enabled	Default: true
<code>imsi</code>	string	International Mobile Subscriber Identity	14-15 digits, unique
<code>custom_attributes</code>	map	Custom key-value data	Optional
<code>sim_id</code>	bigint	Foreign key to SIM	Optional
<code>key_set_id</code>	bigint	Foreign key to Key Set	Required
<code>epc_profile_id</code>	bigint	Foreign key to EPC Profile	Required
<code>ims_profile_id</code>	bigint	Foreign key to IMS Profile	Optional
<code>roaming_profile_id</code>	bigint	Foreign key to Roaming Profile	Optional
<code>subscriber_state_id</code>	bigint	Foreign key to Subscriber State	Auto-created

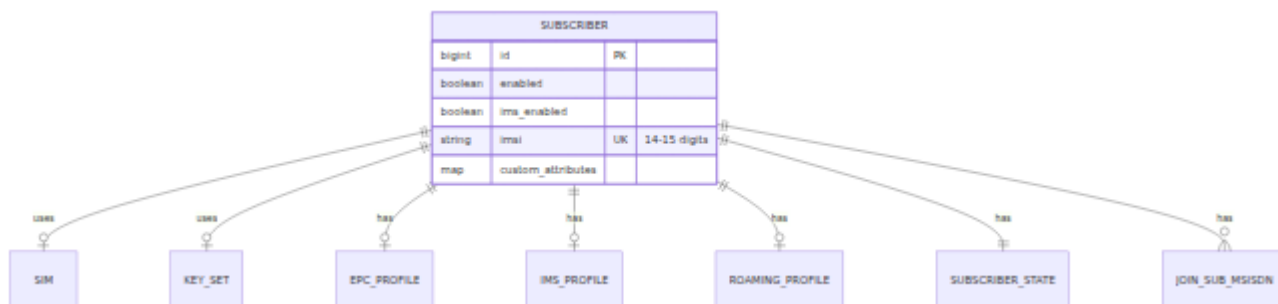
### Key Points:

- Each subscriber must have exactly one IMSI
- IMSI must be 14-15 digits (no letters or special characters)
- A subscriber can have multiple MSISDNs (phone numbers)

- Subscriber state is automatically created when subscriber is created
- `enabled` flag controls all services (data and IMS)
- `ims_enabled` flag controls only IMS services

## SIM

The **SIM** entity represents a physical or embedded SIM card.



### Fields:

Field	Type	Description	Security Level
<code>iccid</code>	string	Integrated Circuit Card ID	Public
<code>sim_vendor</code>	string	SIM manufacturer	Public
<code>batch_name</code>	string	Manufacturing batch	Public
<code>is_esim</code>	boolean	Embedded SIM flag	Public
<code>pin1</code> , <code>pin2</code>	string	PIN codes	Sensitive
<code>puk1</code> , <code>puk2</code>	string	PUK codes	Sensitive
<code>adm1</code> - <code>adm10</code>	string	Administrative codes	Highly Sensitive
<code>kic</code> , <code>kid</code>	binary	OTA security keys	Highly Sensitive

### Key Points:

- ICCID uniquely identifies the SIM card

- One SIM can be assigned to one subscriber at a time
- PIN/PUK codes are for end-user SIM locking
- ADM codes are for administrative SIM operations
- KIC/KID are for SIM OTA (Over-The-Air) updates

## Key Set

The **Key Set** contains cryptographic keys for authentication.

KEY_SET			
bigint	id	PK	
binary	ki		128-bit
binary	opc		128-bit
binary	op		128-bit
binary	amf		16-bit
bigint	sqn		48-bit sequence
string	authentication_algorithm		

used by:



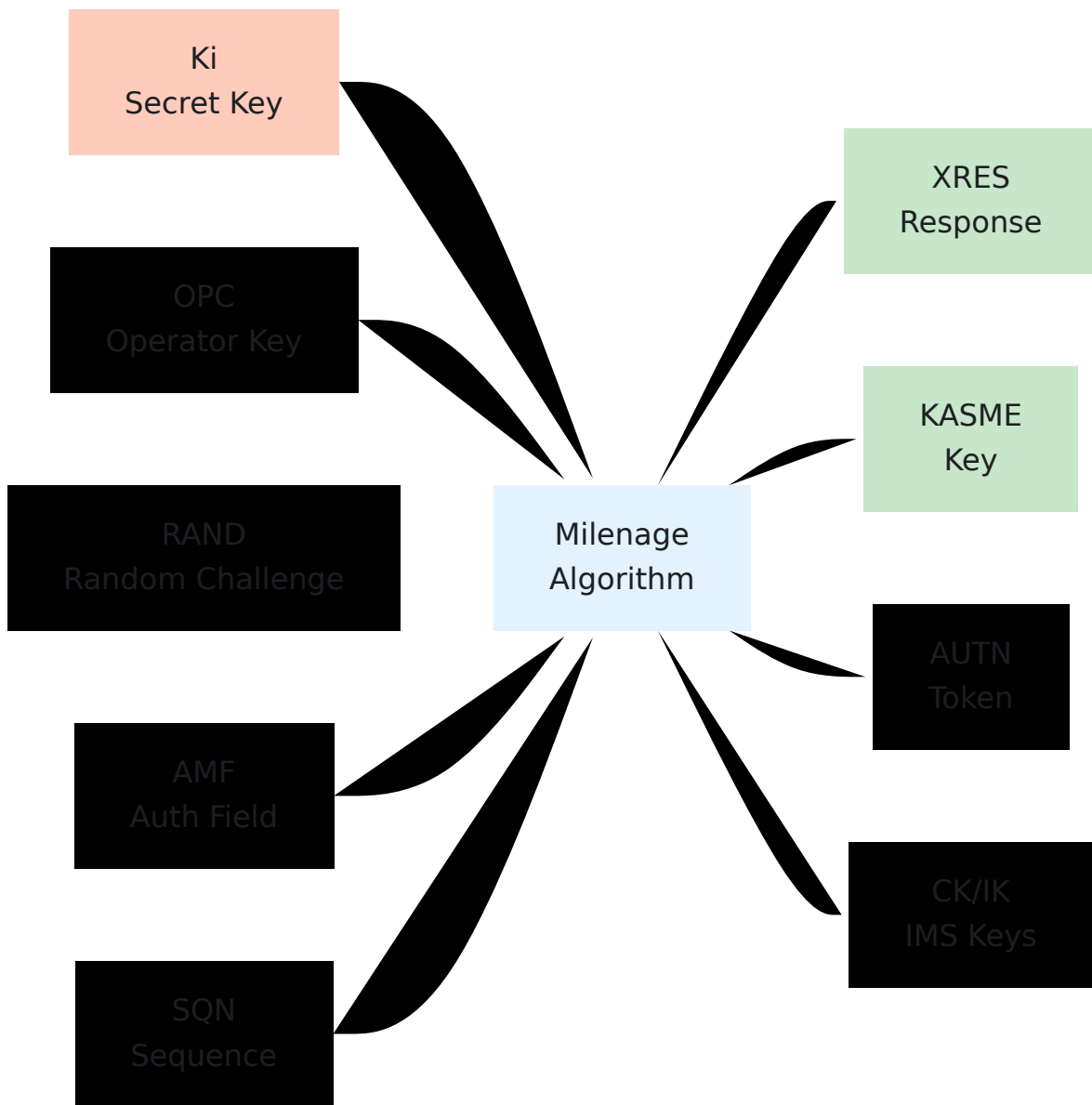
**Fields:**

Field	Type	Description	Size
ki	binary	Secret key	128 bits (16 bytes)
opc	binary	Operator variant key (derived)	128 bits
op	binary	Operator key (for deriving OPC)	128 bits
amf	binary	Authentication Management Field	16 bits (2 bytes)
sqn	bigint	Sequence number (anti-replay)	48 bits
authentication_algorithm	string	Algorithm name	Currently "milenage"
ota_counter	bigint	OTA operation counter	Integer

### Key Points:

- Multiple subscribers can share the same key set
- Ki is the master secret shared with the SIM
- Either OPC or OP must be provided (OPC can be derived from OP)
- SQN is incremented with each authentication
- Milenage is currently the only supported algorithm

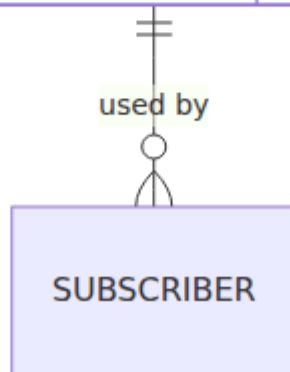
### Authentication Algorithm:



## MSISDN

The **MSISDN** represents a phone number.

KEY_SET			
bigint	id	PK	
binary	ki		128-bit
binary	opc		128-bit
binary	op		128-bit
binary	amf		16-bit
bigint	sqn		48-bit sequence
string	authentication_algorithm		



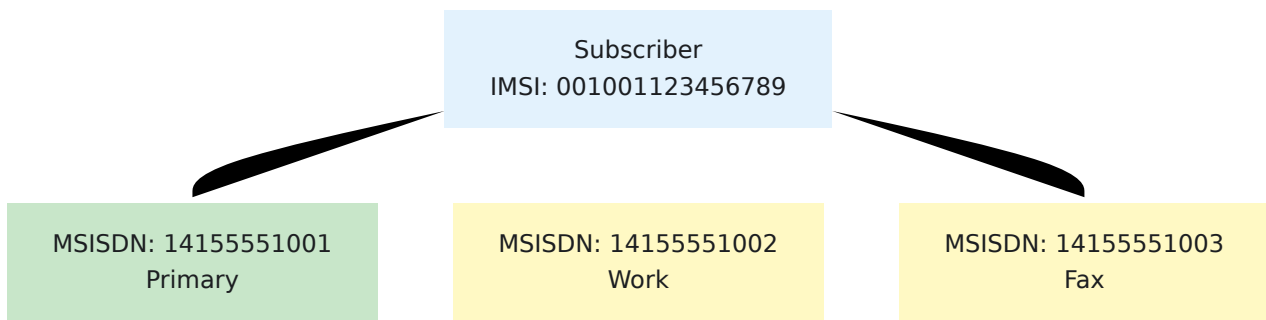
### Fields:

Field	Type	Description	Format
msisdn	string	Mobile Station ISDN Number	1-15 digits, E.164 format

### Key Points:

- MSISDN is the phone number in international format
- Multiple MSISDNs can be assigned to one subscriber
- One MSISDN cannot be shared between multiple subscribers
- Format: Country code + National number, digits only — no + prefix (e.g., 14155551234 not +14155551234)
- The + prefix is stripped automatically during Diameter Cx/Rx lookups when the S-CSCF or P-CSCF sends a SIP URI with or without the +

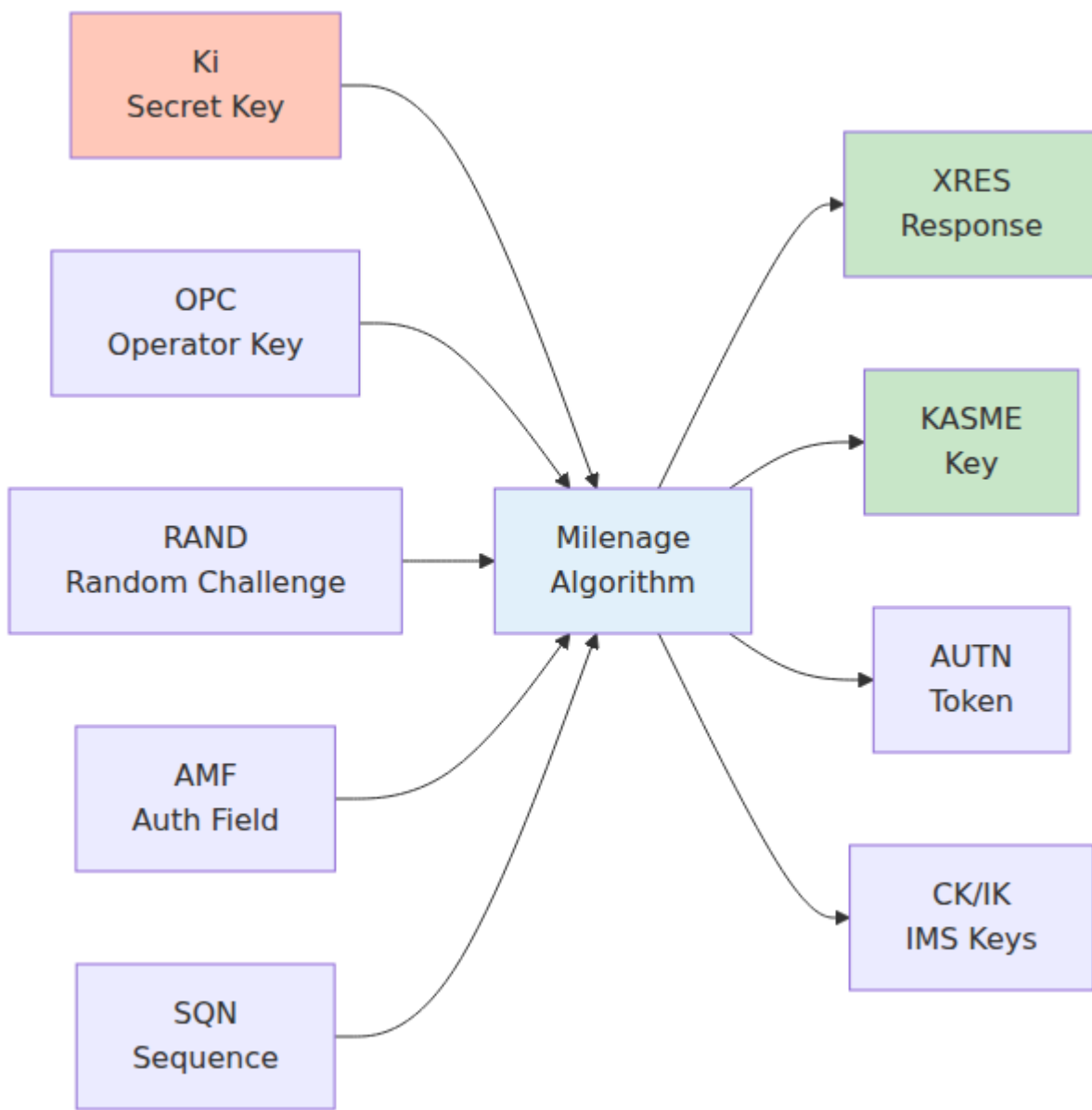
## Multi-MSISDN Pattern:



# Profile Entities

## EPC Profile

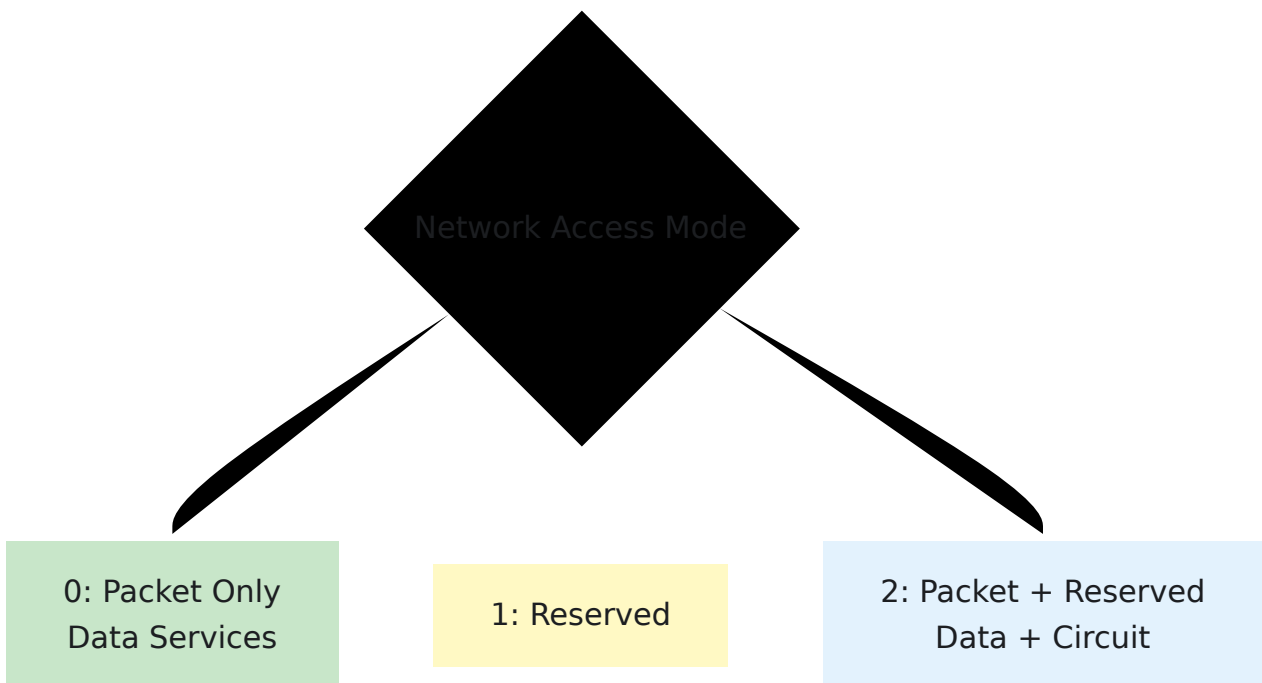
The **EPC Profile** defines data service characteristics for LTE.



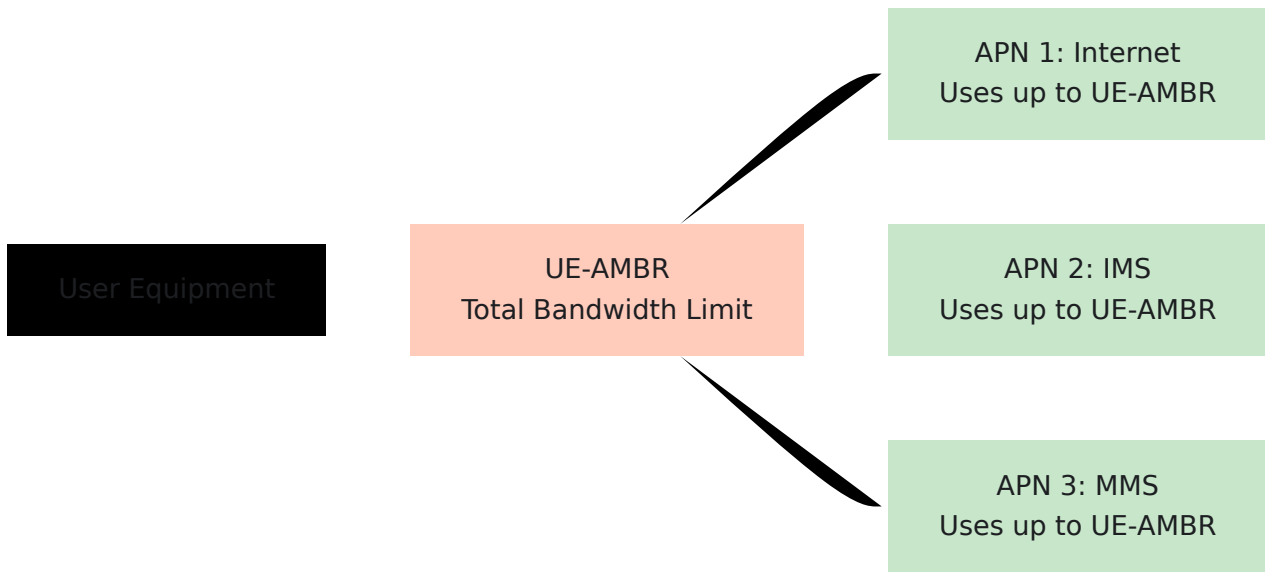
**Fields:**

Field	Type	Description	
name	string	Profile name	Text
ue_ambr_dl_kbps	integer	Download bandwidth limit	Kbps
ue_ambr_ul_kbps	integer	Upload bandwidth limit	Kbps
network_access_mode	string	Access restrictions	"packe "packe
tracking_area_update_interval_seconds	integer	TAU interval	Second

**Network Access Modes:**

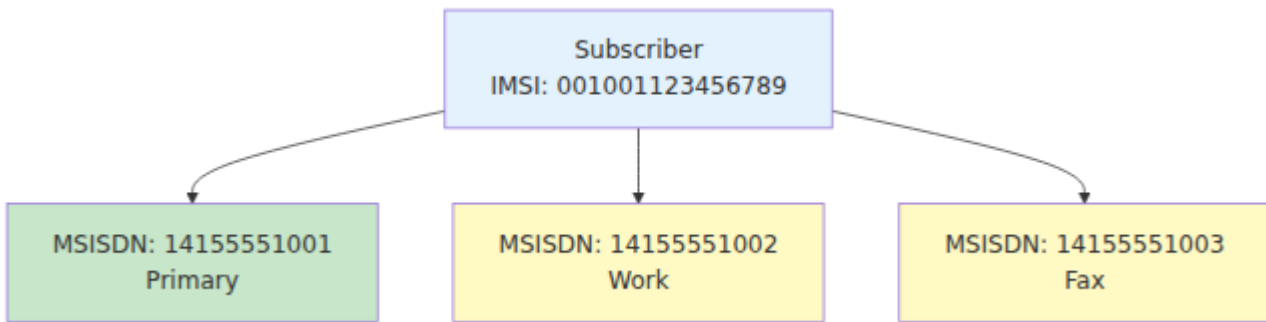


**AMBR (Aggregate Maximum Bit Rate):**



## IMS Profile

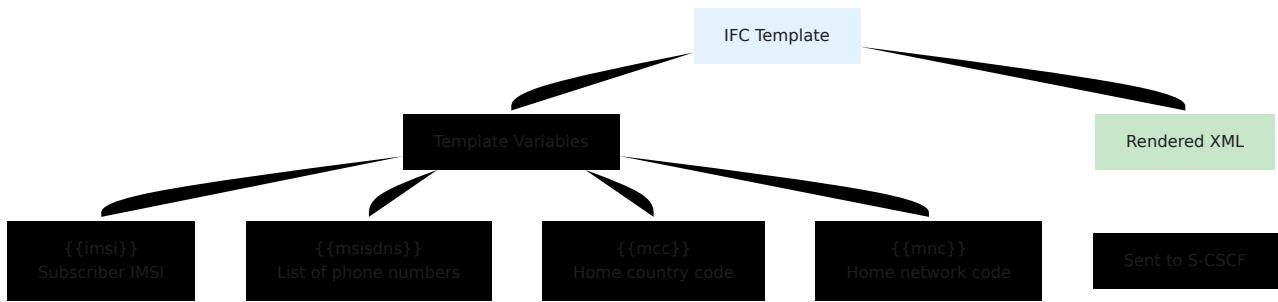
The **IMS Profile** defines voice/video service characteristics.



### Fields:

Field	Type	Description	Format
<code>name</code>	string	Profile name	Text
<code>ifc_template</code>	text	Initial Filter Criteria XML template	XML with variables

### IFC Template Variables:

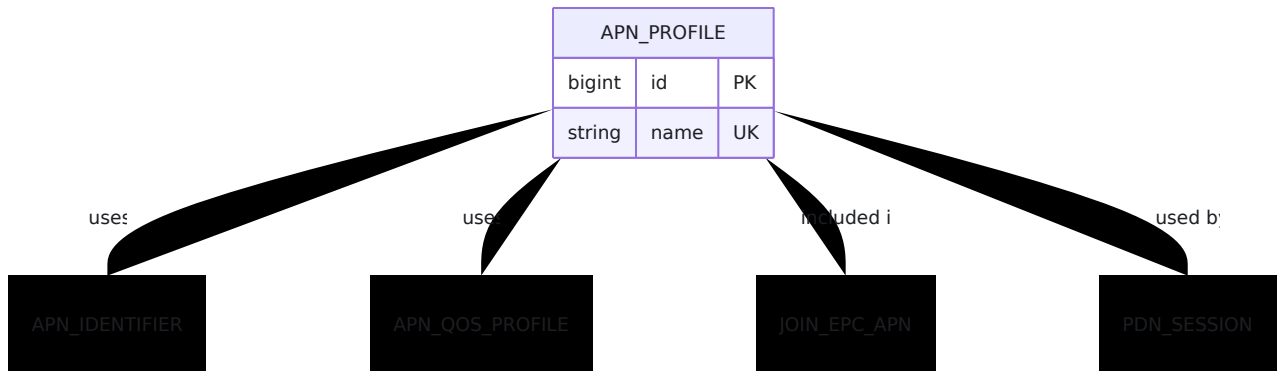


### Key Points:

- IFC (Initial Filter Criteria) controls call routing in IMS
- Template is rendered when subscriber registers
- Variables are substituted with actual subscriber data
- Sent to S-CSCF during IMS registration

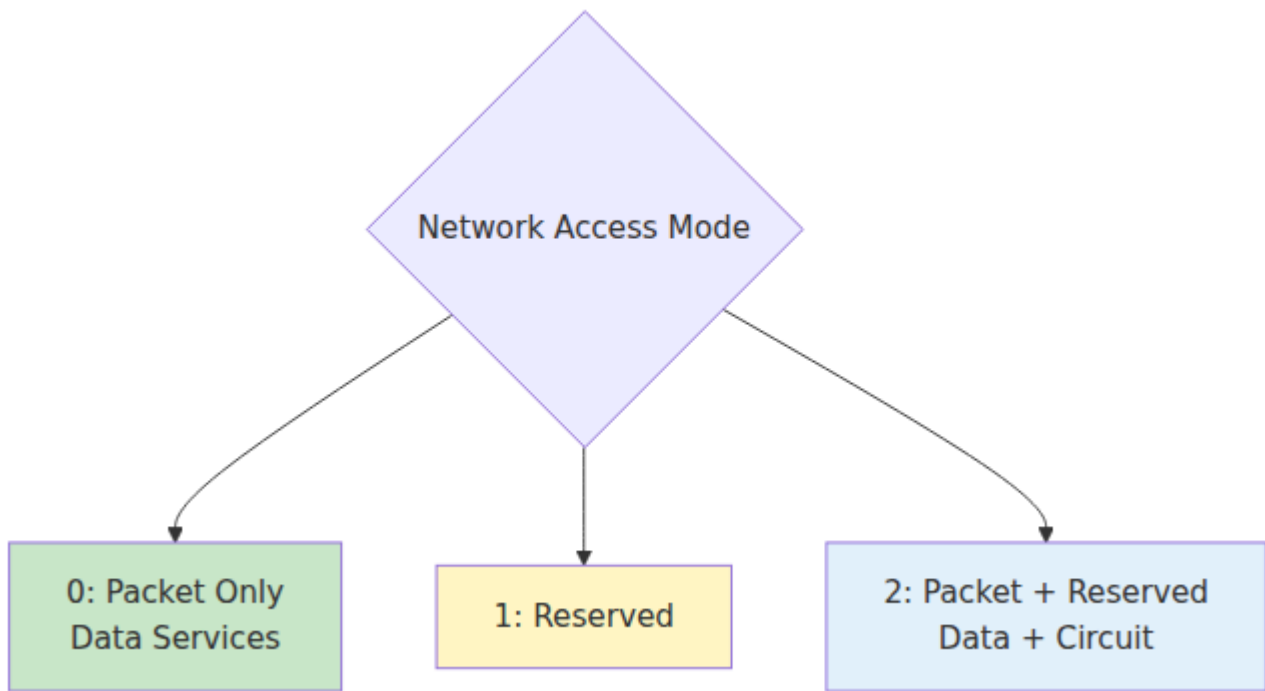
## APN Profile

The **APN Profile** defines characteristics for a specific data access point.



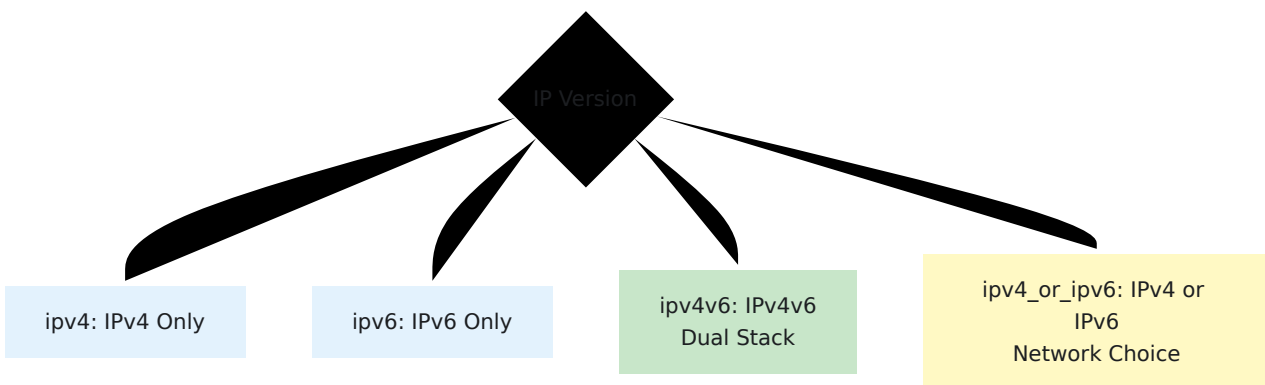
### Related Entities:

#### APN Identifier

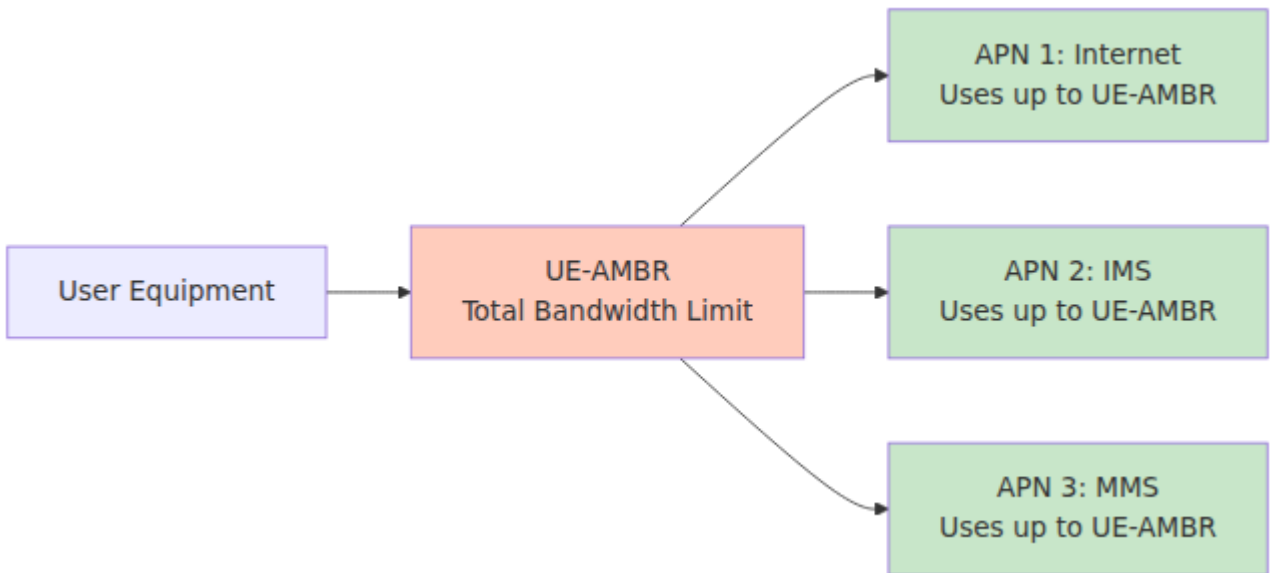


Field	Type	Description	Example
apn	string	APN name	"internet", "ims", "mms"
ip_version	string	IP protocol support	See below

**IP Version Options:**



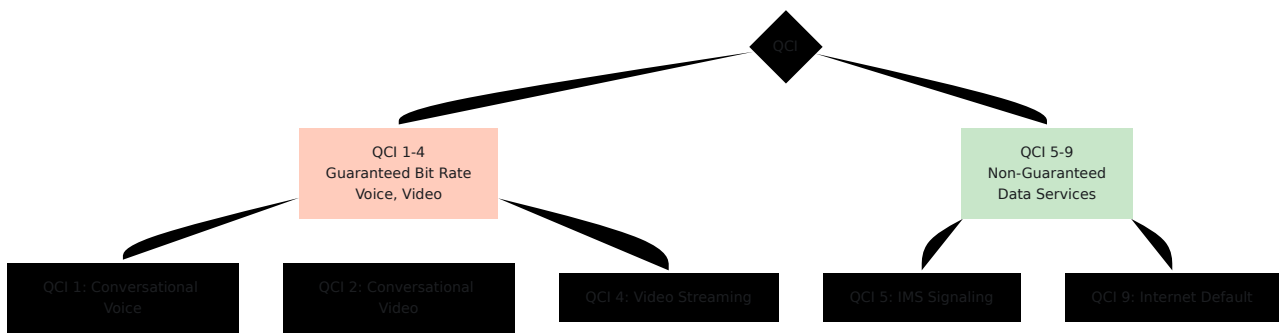
**APN QoS Profile**



### QoS Parameters:

Parameter	Description	Range	Default Bearer
<code>qci</code>	QoS Class Identifier	1-9	QCI 9 (Internet)
<code>allocation_retention_priority</code>	ARP priority	1-15	8 (lower priority)
<code>apn_ambr_dl_kbps</code>	APN download limit	0+	Varies
<code>apn_ambr_ul_kbps</code>	APN upload limit	0+	Varies
<code>pre_emption_capability</code>	Can preempt others	true/false	false
<code>pre_emption_vulnerability</code>	Can be preempted	true/false	true

### QCI Values:



## Roaming Profile

The **Roaming Profile** controls access when subscriber visits other networks.

ROAMING_PROFILE			
bigint	id	PK	
string	name	UK	
string	data_action_if_no_rules_match		allow or deny
string	ims_action_if_no_rules_match		allow or deny

include



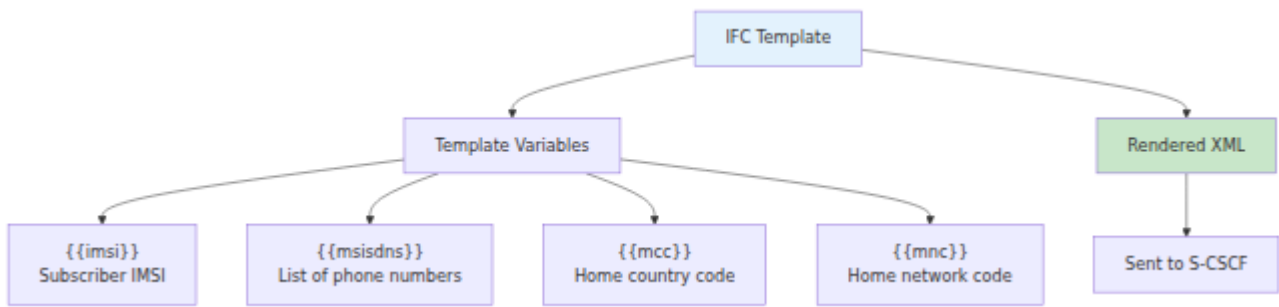
assigned to



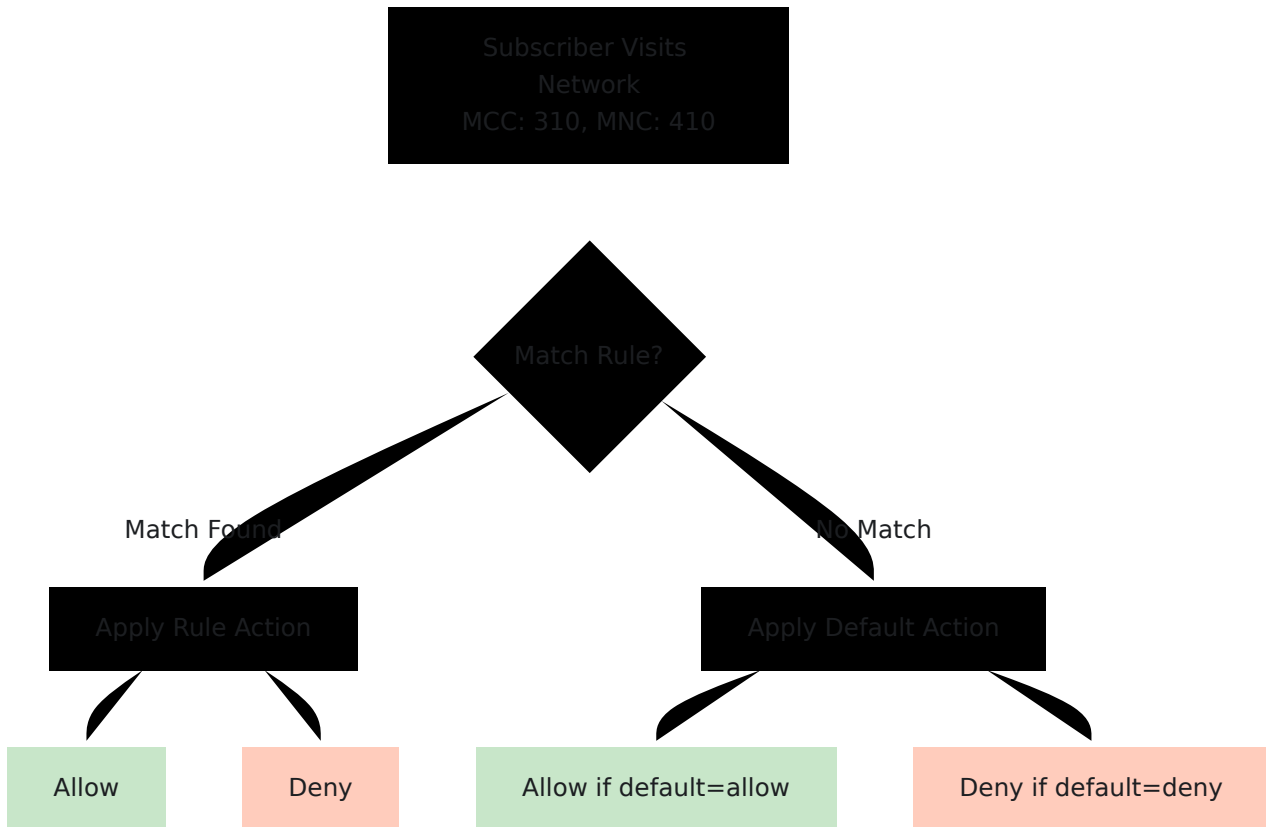
reference



**Roaming Rule:**



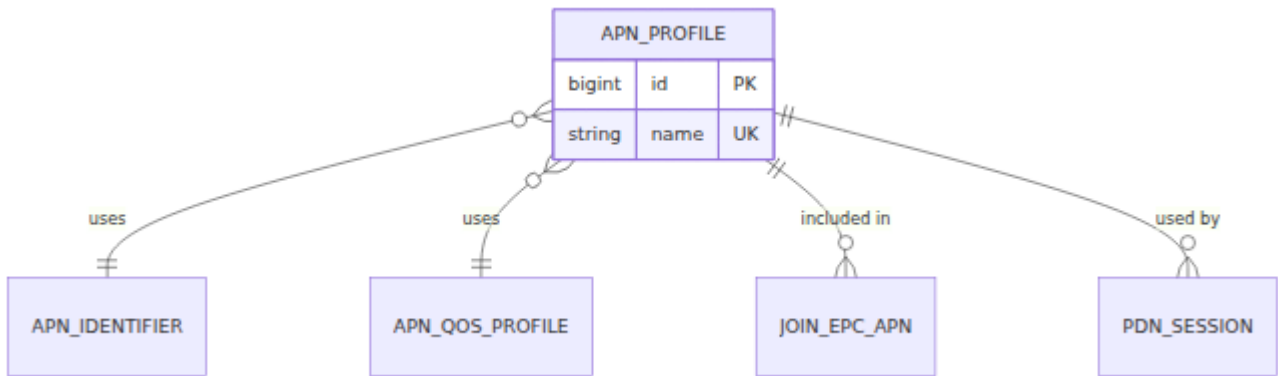
## Rule Evaluation:



# State Entities

## Subscriber State

The **Subscriber State** tracks real-time subscriber status.



## Key Fields:

## Location Information:

- `last_seen_mcc`, `last_seen_mnc` - Visited network
- `last_seen_tac` - Tracking Area Code
- `last_seen_cell_id` - Cell ID
- `last_seen_enodeb_id` - eNodeB ID
- `last_seen_eci` - E-UTRAN Cell Identifier

## Network Elements:

- `last_seen_mme` - Current MME serving subscriber
- `last_seen_realm` - Diameter realm of MME
- `last_seen_rat_type` - Radio Access Technology (LTE, 5G, etc.)

## IMS Information:

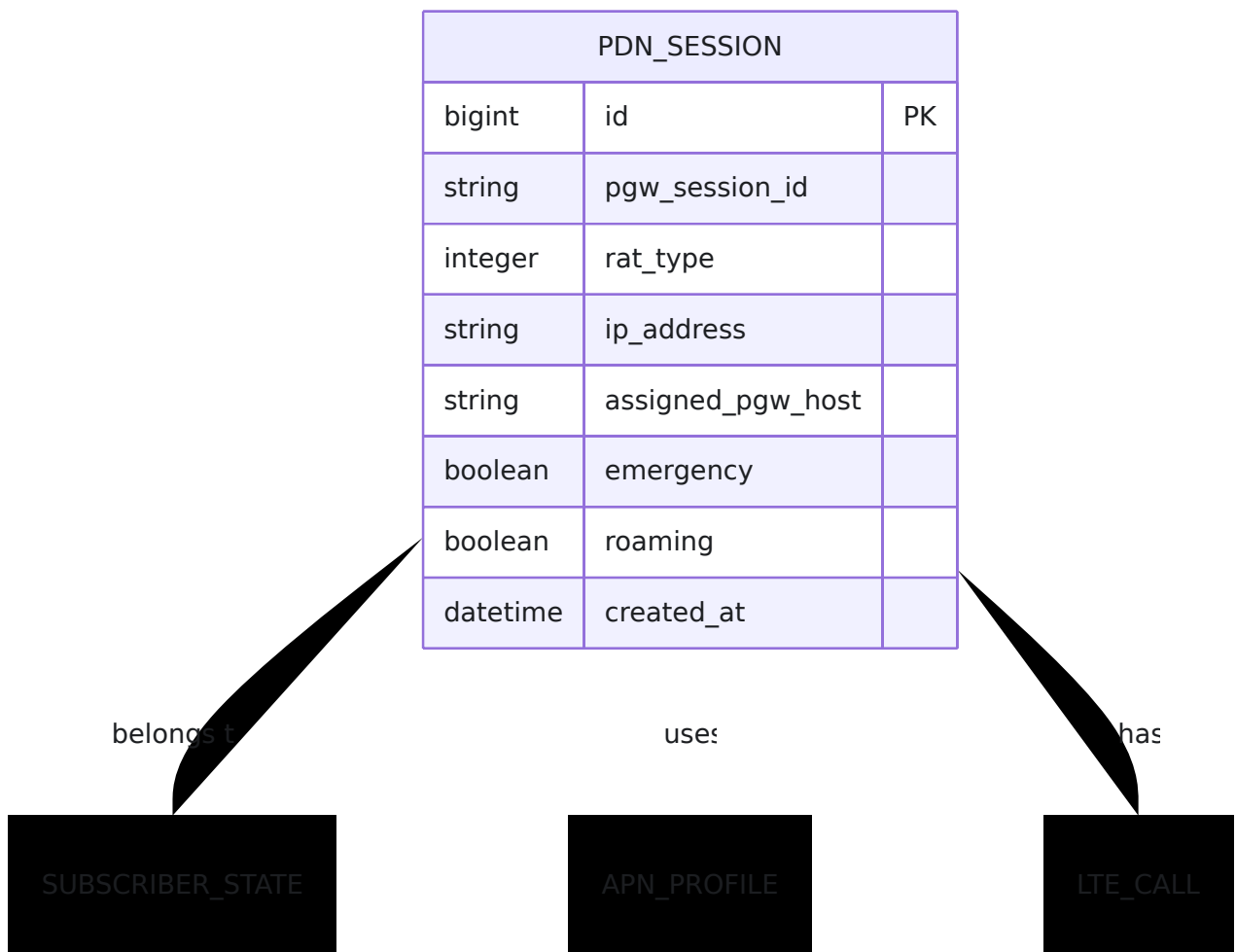
- `assigned_scsf` - Current S-CSCF serving subscriber
- `ims_public_identity` - SIP URI (e.g., `sip:+14155551234@ims.example.com`)
- `sh_repository_data` - Custom IMS profile data

## Timestamps:

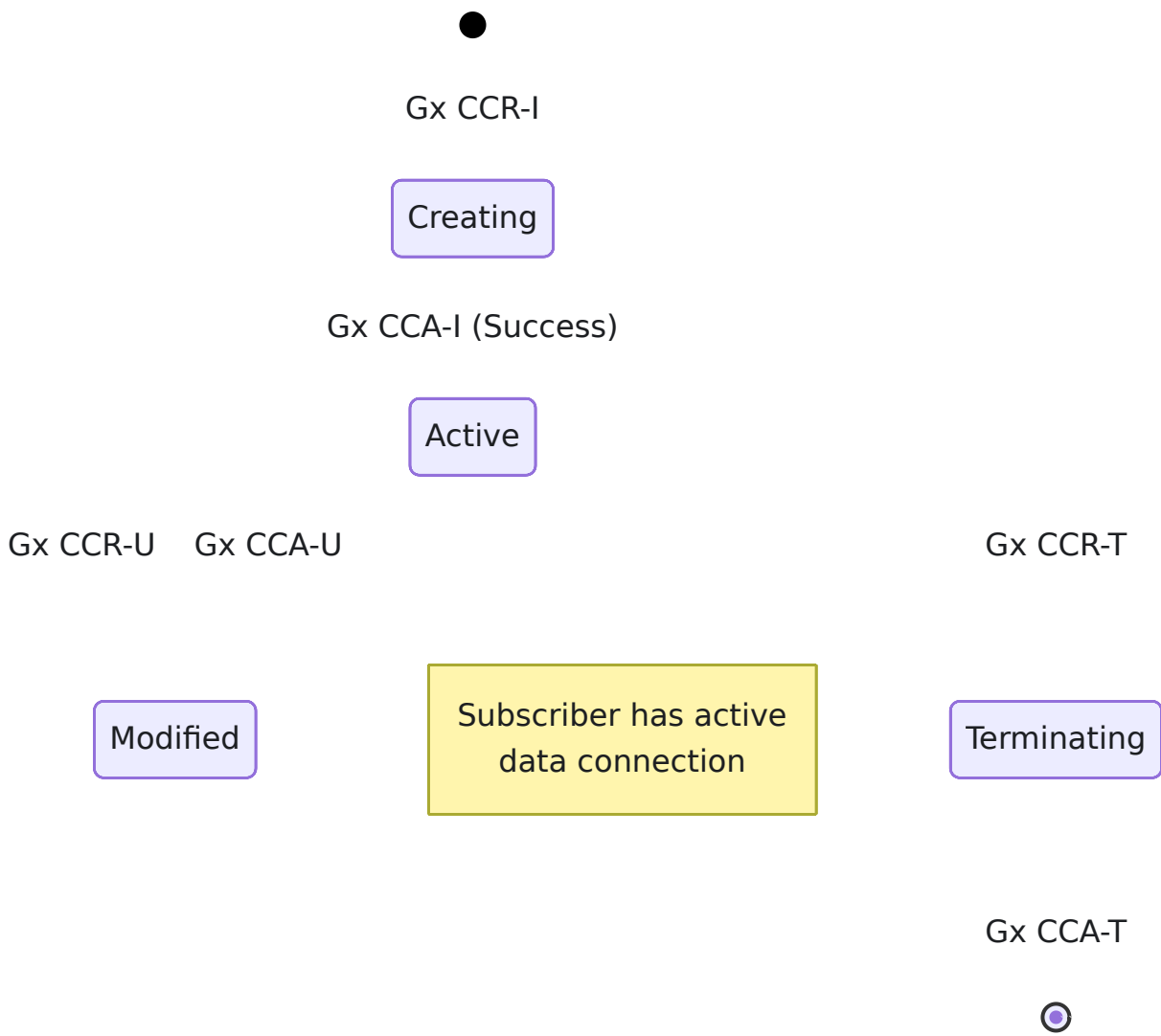
- `last_seen_at` - Last Diameter message received
- Various `last*_at` timestamps for different procedures

# PDN Session

The **PDN Session** represents an active data connection.

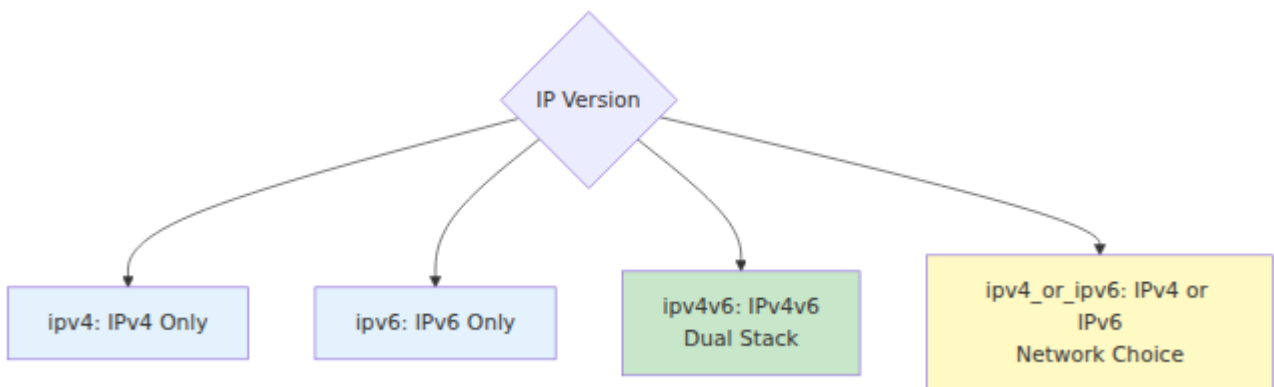


**PDN Session Lifecycle:**

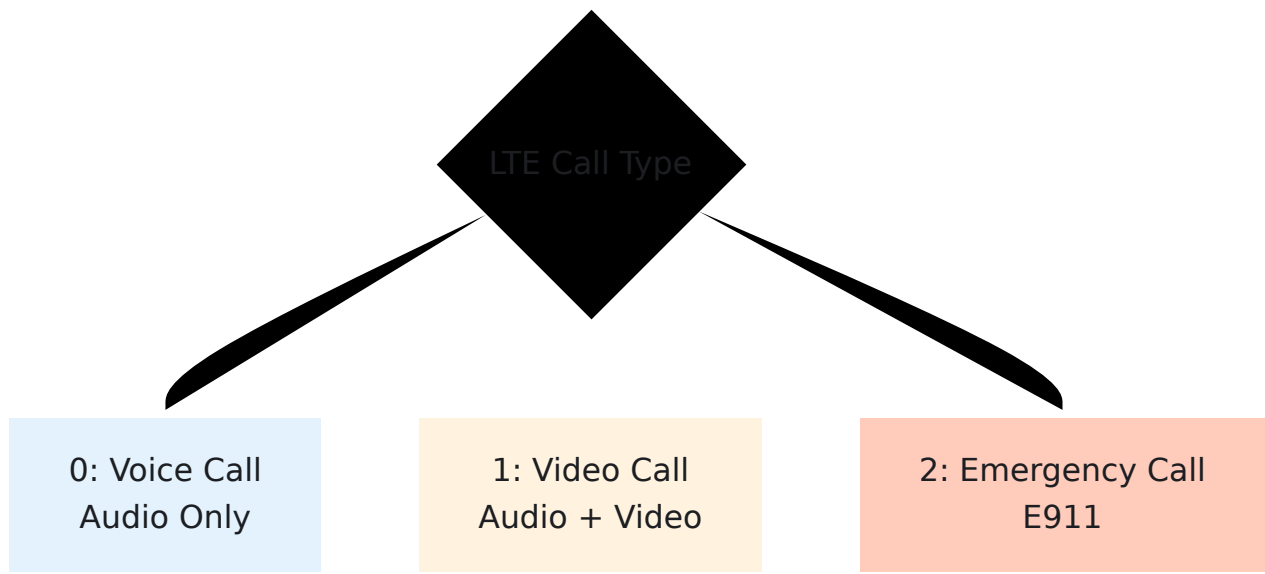


## LTE Call

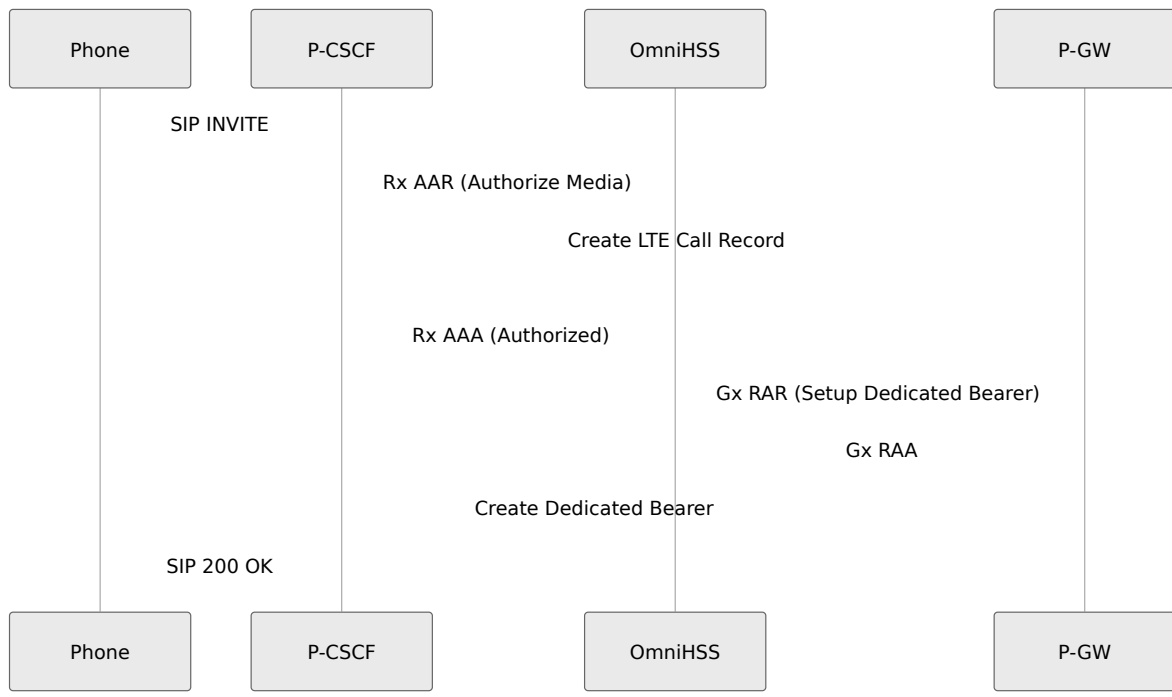
The **LTE Call** represents an active VoLTE voice/video call.



**Call Types:**

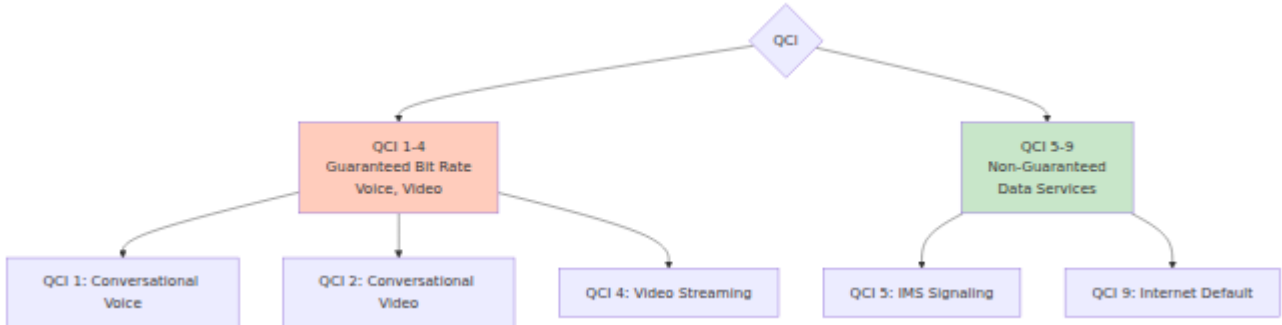


**VoLTE Call Flow:**



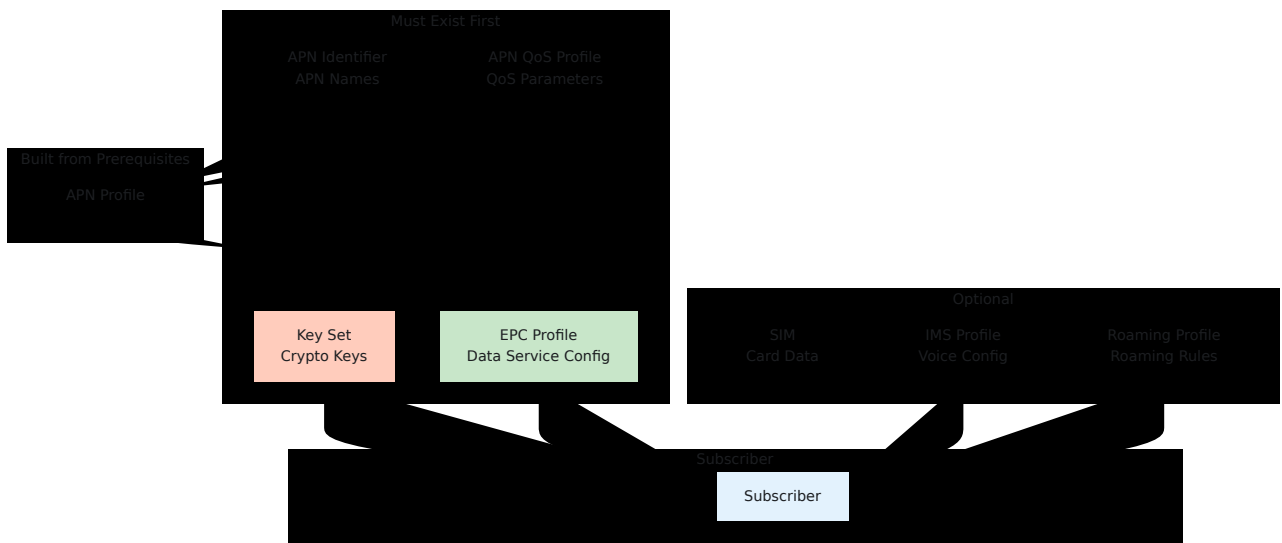
# Entity Relationship Diagrams

## Complete Entity Relationships

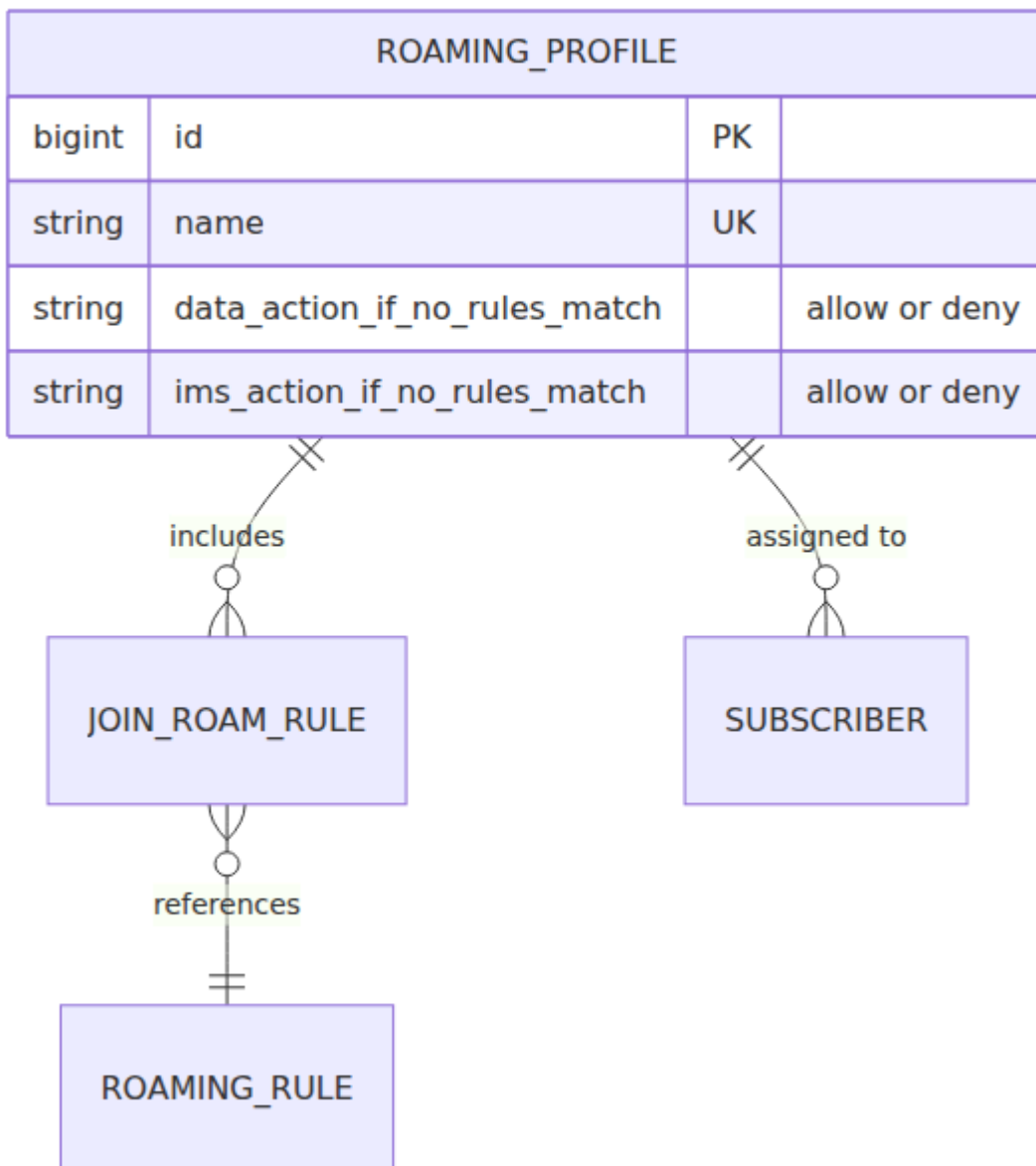


## Provisioning Relationships

This diagram shows what must exist before creating a subscriber:

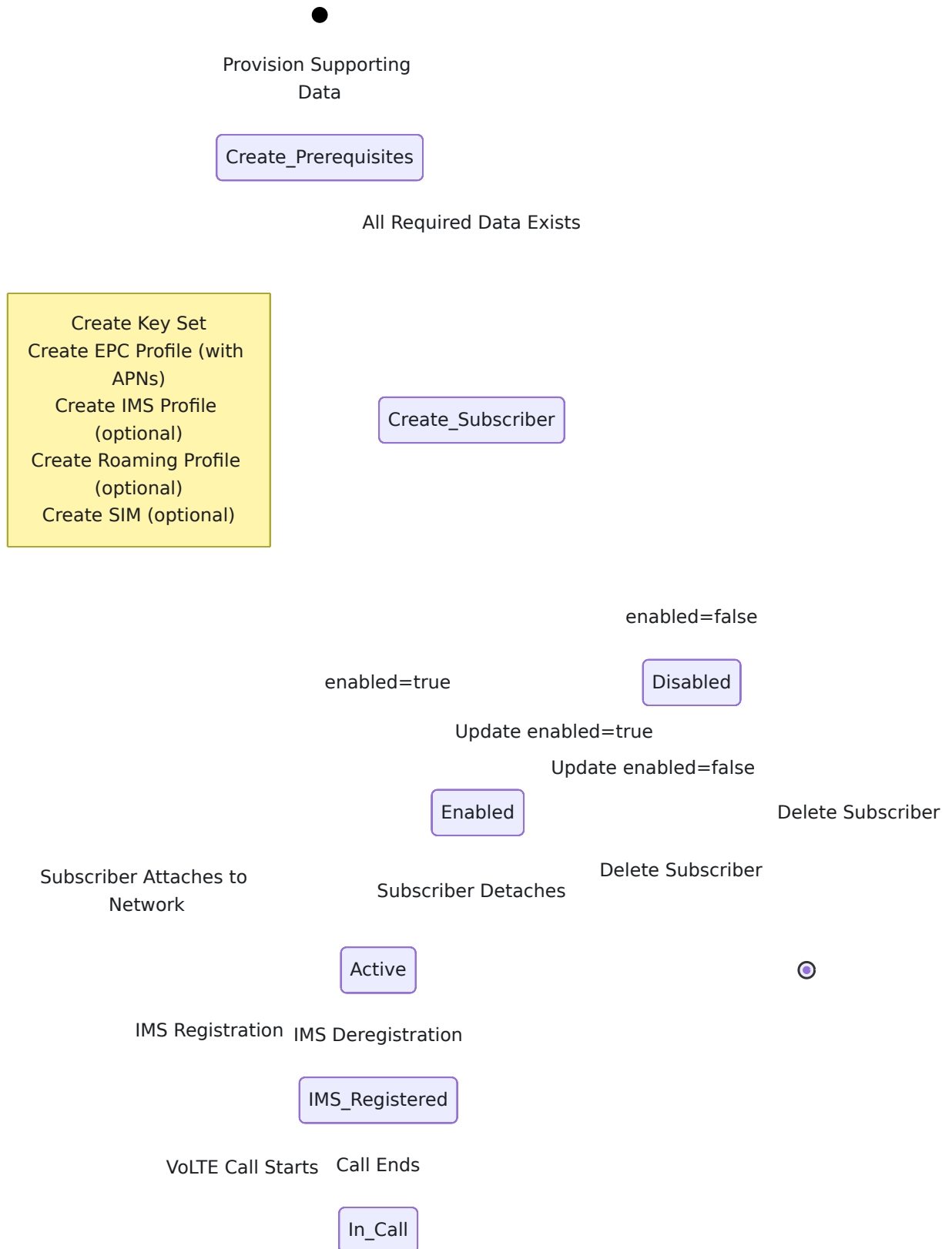


# Session State Relationships



# Entity Lifecycle

## Subscriber Provisioning Lifecycle



# Session Lifecycle



Subscriber Idle

No\_Sessions

Data Connection Starts    Data Connection Ends

PDN\_Active

VoLTE Call Starts

VoLTE Call Ends

PDN Session record  
exists  
in database

PDN\_And\_Call

Second Call Starts    Second Call Ends

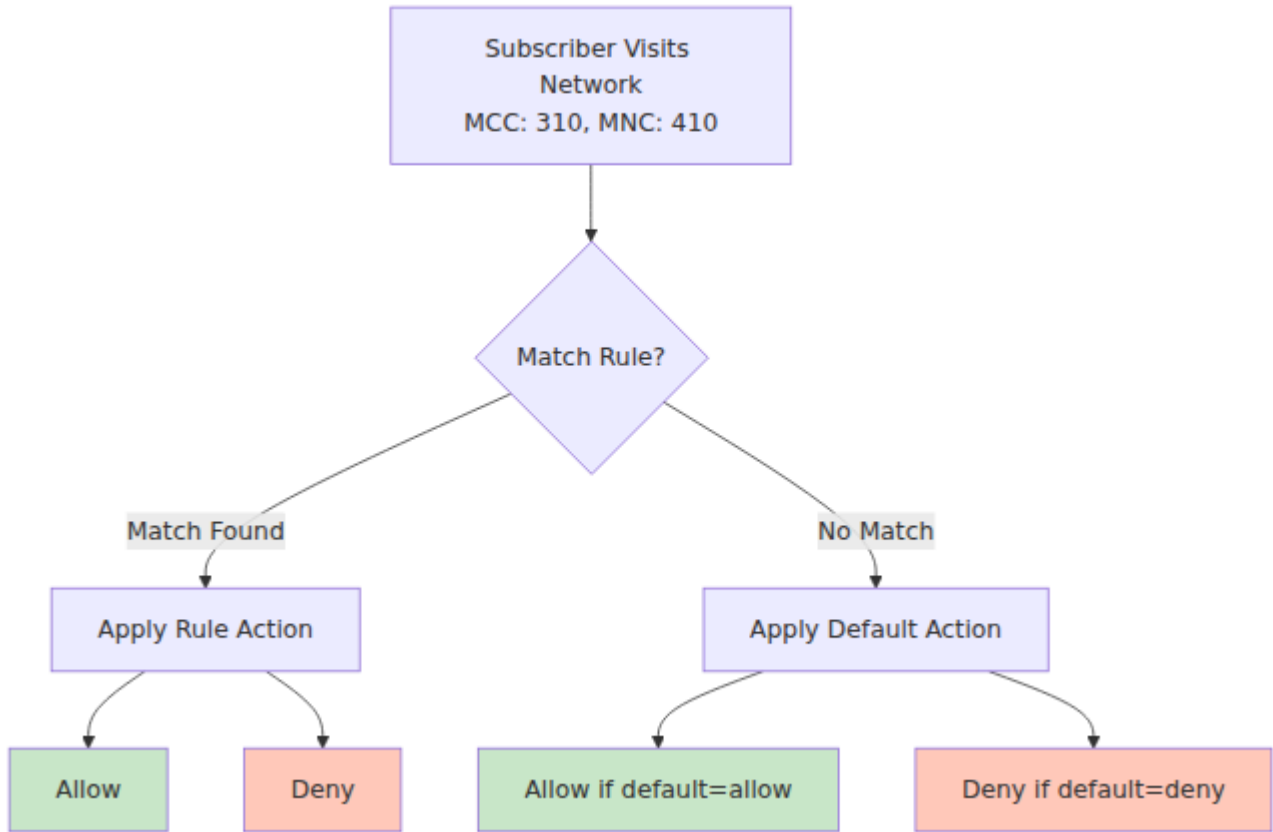
Multiple\_Calls

PDN Session + LTE Call  
records exist

---

# Data Flow Patterns

## Authentication Flow



# Location Update Flow

S6a ULR Request

Lookup Subscriber  
by IMSI

Load EPC Profile  
+ APN Profiles

Update Subscriber State  
Location, MME, etc.

Build Subscription Data  
AMBR, APNs, QoS

S6a ULA Response

# IMS Registration Flow

Cx SAR Request

Lookup Subscriber  
by IMSI/MSISDN

Load IMS Profile  
+ MSISDNs

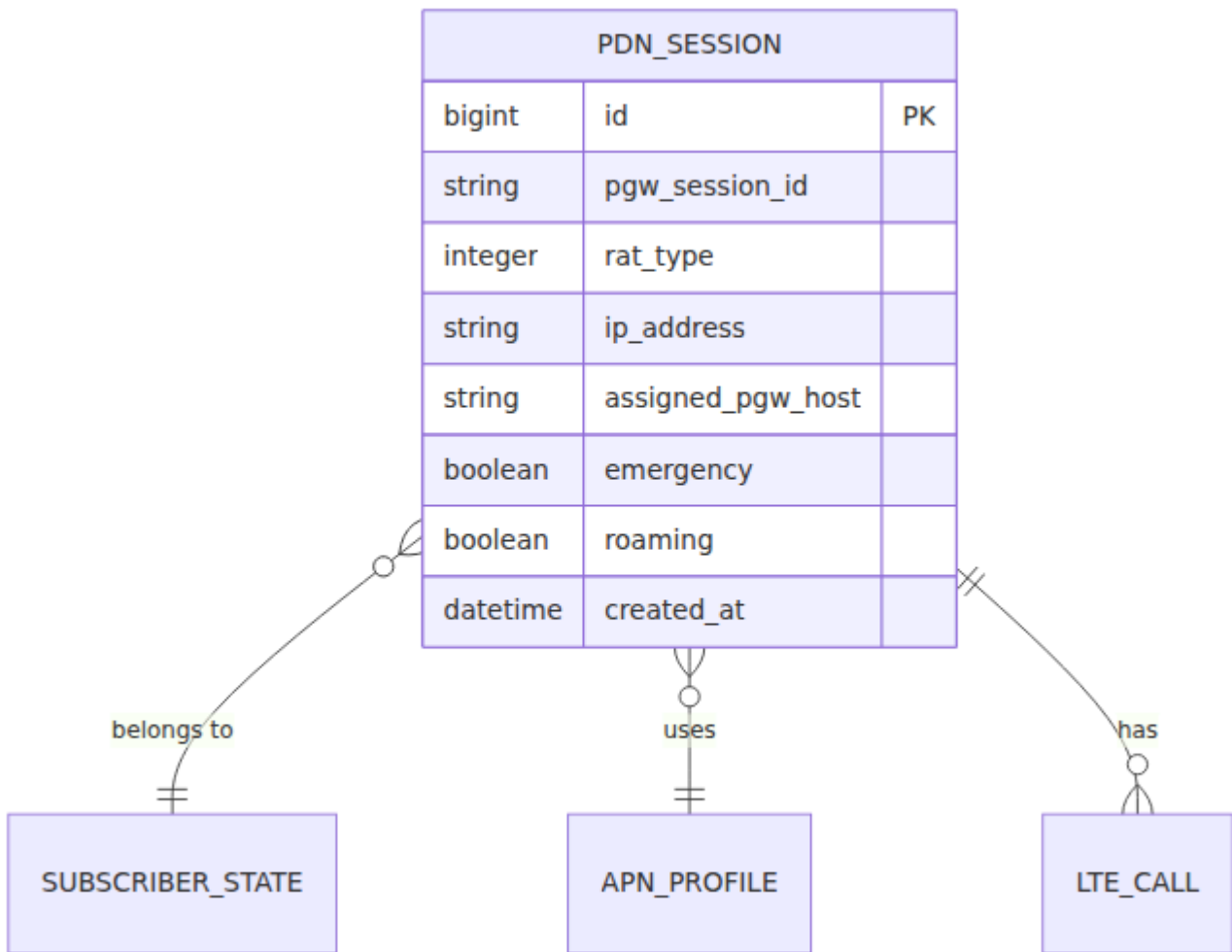
Select S-CSCF  
Random/Round-Robin

Render IFC Template  
with Variables

Update Subscriber State  
S-CSCF Assignment

Cx SAA Response

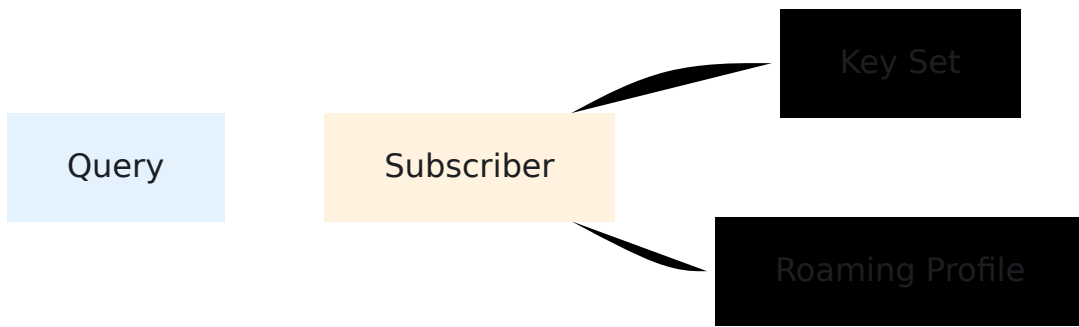
# Session Establishment Flow



# Query Optimization Patterns

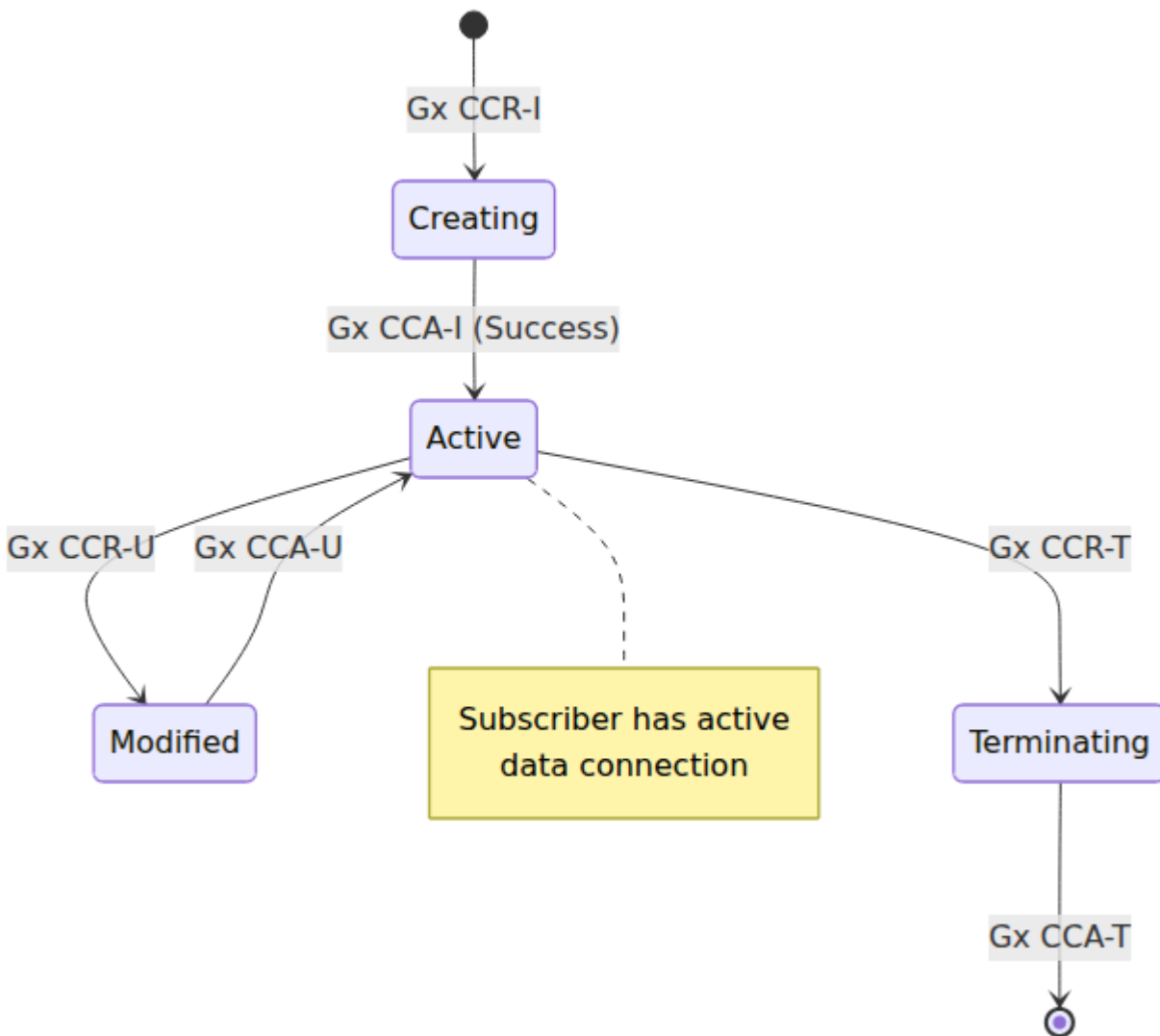
OmniHSS optimizes database queries by selectively preloading only the necessary associations for each operation:

# Minimal Query (Authentication)



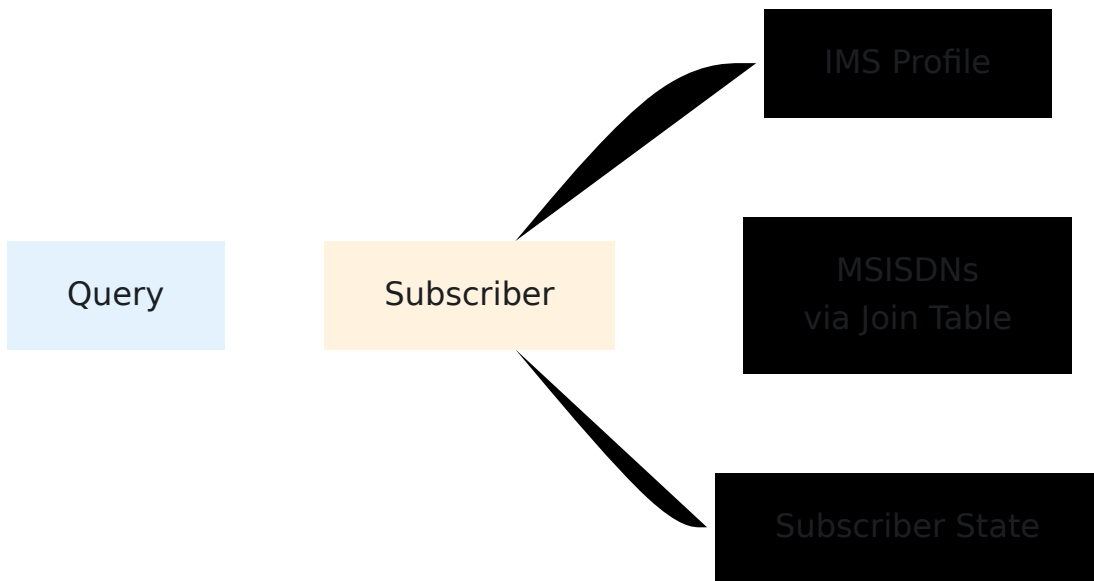
**Use Case:** S6a AIR - Only needs crypto keys and roaming rules

# Moderate Query (Location Update)



**Use Case:** S6a ULR - Needs complete EPC profile data

# Full Query (IMS Registration)



**Use Case:** Cx SAR - Needs IMS profile and all phone numbers

---

[← Back to Operations Guide](#) | [Next: API Reference](#) →

# Diameter Response Data Mapping

[← Back to Operations Guide](#)

This document provides detailed mermaid diagrams showing where each field in Diameter protocol responses is sourced from in the OmniHSS system.

---

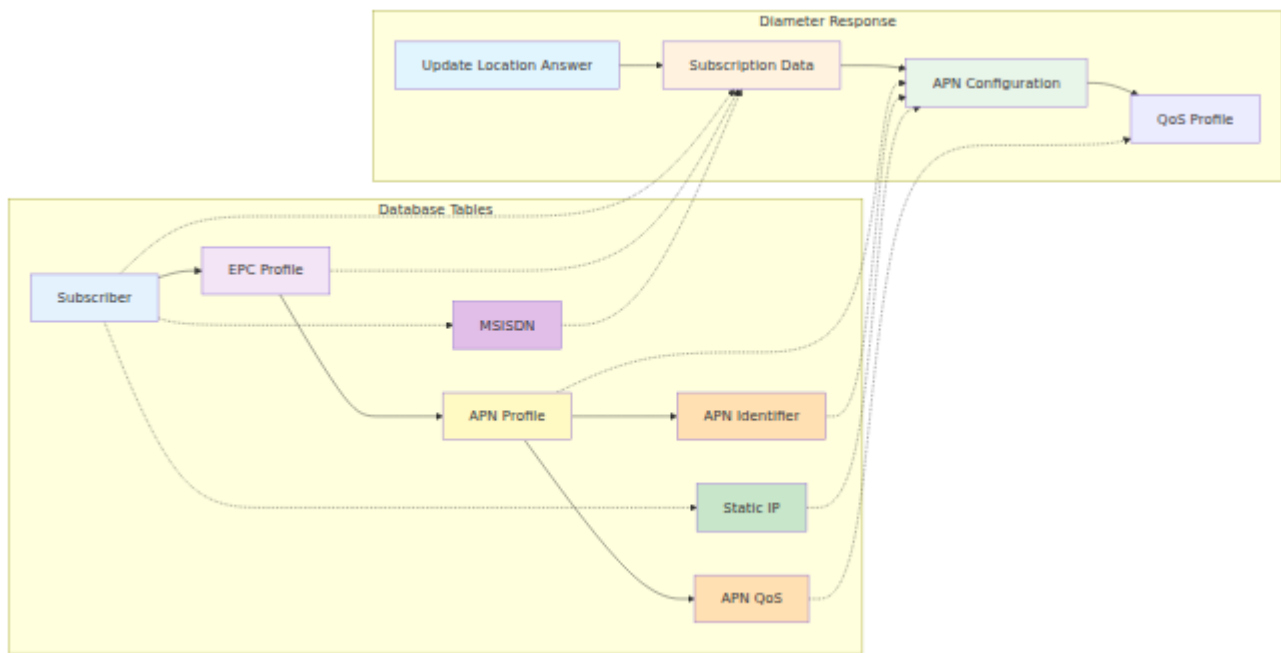
## Table of Contents

- [Update Location Answer \(S6a ULA\)](#)
  - [Authentication Information Answer \(S6a AIA\)](#)
  - [Server Assignment Answer \(Cx SAA\)](#)
  - [Credit Control Answer \(Gx CCA\)](#)
  - [User Data Answer \(Sh UDA\)](#)
  - [ME Identity Check Answer \(S13 ECA\)](#)
- 

## Update Location Answer (S6a ULA)

The Update Location Answer is sent by the HSS to the MME during LTE attach procedures. This diagram shows the complete data flow from database tables to Diameter AVPs.

# Data Source Mapping



## Detailed Field Mapping

Database Source	Field	D
<b>subscriber.enabled</b>	true/false	Su St
<b>msisdn.msisdn</b>	'14155551234'	MS
<b>epc_profile.ue_ambr_ul_kbps</b>	50000	Ma Re Ba UL
<b>epc_profile.ue_ambr_dl_kbps</b>	100000	Ma Re Ba DL
<b>epc_profile.network_access_mode</b>	'packet_only'	Ne Ac Mo
<b>apn_identifier.apn</b>	'internet'	Se Se
<b>apn_identifier.ip_version</b>	'ipv4v6'	PD
<b>apn_qos_profile.qci</b>	9	Qc Id
<b>apn_qos_profile.allocation_retention_priority</b>	8	Pr Le

Database Source	Field	D
apn_qos_profile.pre_emption_capability	false	Pr en Ca
apn_qos_profile.pre_emption_vulnerability	true	Pr en Vu
apn_qos_profile.apn_ambr_ul_kbps	25000	AF UL
apn_qos_profile.apn_ambr_dl_kbps	50000	AF DL
static_ip.ipv4_static_ip	'100.64.1.1'	Se Pa Ac (IF
static_ip.ipv6_static_ip	'2606:4700::1111'	Se Pa Ac (IF

### Key Transformations:

- AMBR bandwidth:** Database stores in kbps, Diameter expects bps (multiply by 1000)
- IP Version encoding:** 0=IPv4, 1=IPv6, 2=IPv4v6, 3=IPv4\_or\_IPv6
- Subscriber Status:** enabled: true → 0 (SERVICE\_GRANTED), enabled: false → 1 (OPERATOR\_DETERMINED\_BARRING)
- Context-Identifier:** Sequential numbering (0, 1, 2...) for each APN in profile

- 5. **Static IP:** Only included if assigned via `static_ips` many-to-many relationship

### Business Logic Validation:

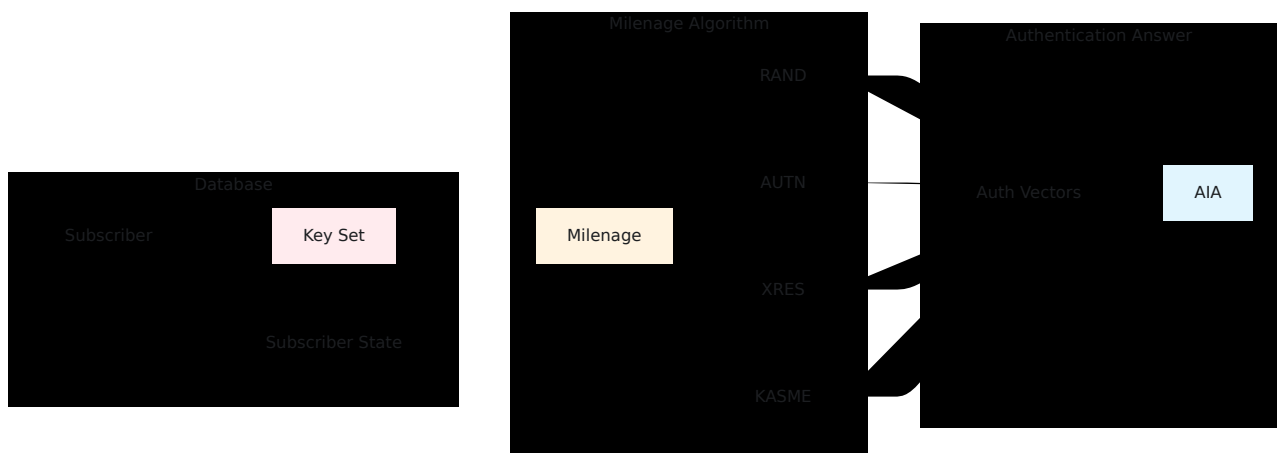
- Roaming check: Match visited PLMN against `roaming_profile.roaming_rules`
- Subscriber enabled check: `subscriber.enabled == true`
- Filter APNs: May exclude IMS APNs if roaming policy denies IMS

---

# Authentication Information Answer (S6a AIA)

The Authentication Information Answer provides authentication vectors for LTE/EPC subscribers.

## Data Source Mapping



### Key Components:

1. **Cryptographic Keys:** All keys stored as hex strings in `key_set` table
2. **SN Management:** Sequence number incremented after each auth vector generation (prevents replay attacks)
3. **Milenage Algorithm:** 3GPP TS 35.206 - generates authentication vectors
4. **KASME Derivation:** Key derived from CK||IK using KDF per TS 33.401

## Security Features:

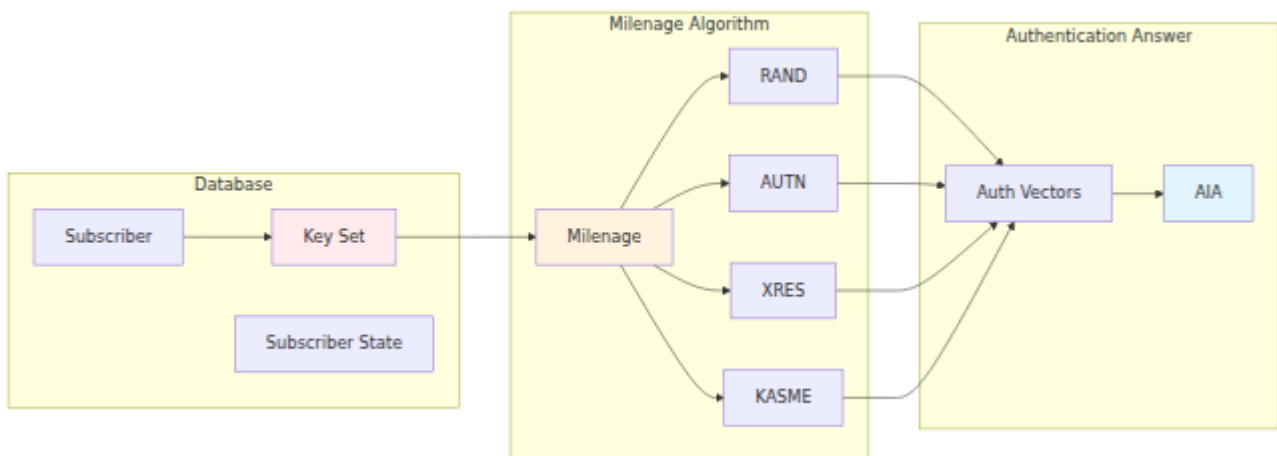
- SQN stored per subscriber (not global)
- Ki/OPc never leave HSS (only derived values transmitted)
- AUTN includes sequence number (SQN) and AMF for network authentication
- Milenage algorithm provides mutual authentication between UE and network

---

# Server Assignment Answer (Cx SAA)

The Server Assignment Answer is sent by the HSS to the S-CSCF during IMS registration.

## Data Source Mapping



## Key Features:

1. **IFC Template:** XML template stored in `ims_profile.ifc_template`
2. **Dynamic Substitution:** Replaces `{{msisdn}}`, `{{imsi}}`, `{{impu}}` at runtime
3. **S-CSCF Assignment:** Stores assigned S-CSCF in `subscriber_state.assigned_scscf`
4. **IMS Public Identity:** Format: `sip:+{msisdn}@{ims_domain}` or `tel:+{msisdn}`

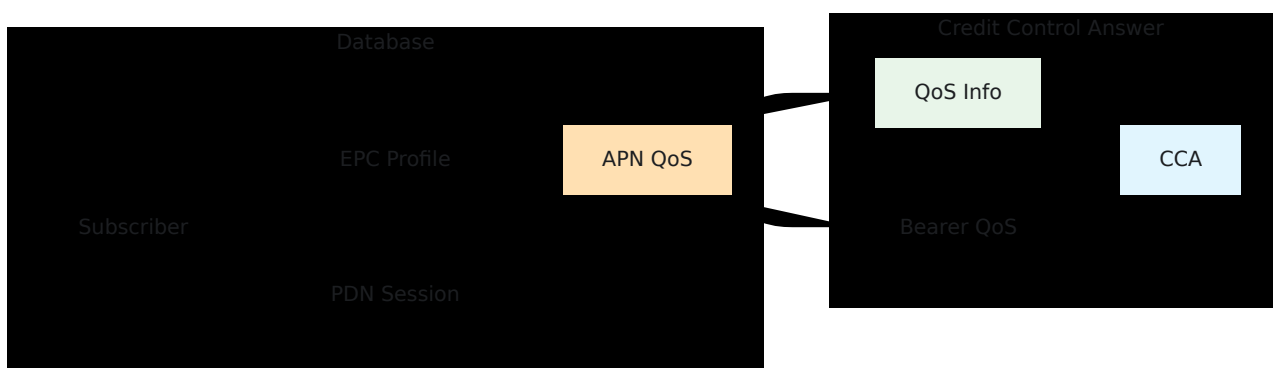
## IFC Template Parameters:

- `{{msisdn}}` - First MSISDN from subscriber
  - `{{imsi}}` - Subscriber IMSI
  - `{{impu}}` - IMS Public User Identity (from subscriber\_state)
  - `{{impi}}` - IMS Private User Identity (typically IMSI@realm)
- 

# Credit Control Answer (Gx CCA)

The Credit Control Answer is sent by the PCRF function to the PGW during bearer establishment.

## Data Source Mapping



## Key Features:

1. **Session Tracking:** Creates/updates `pdn_session` record for each bearer
2. **QoS Enforcement:** Provides QCI and bandwidth limits from APN QoS profile
3. **Charging Rules:** Returns default charging rules for billing integration
4. **CC-Request-Type:** Handles INITIAL (1), UPDATE (2), TERMINATION (3)

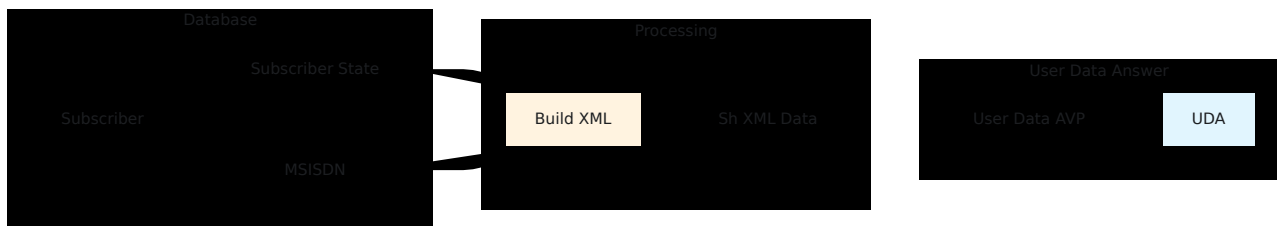
## Session State Management:

- `INITIAL_REQUEST`: Creates new PDN session record
  - `UPDATE_REQUEST`: Updates existing PDN session
  - `TERMINATION_REQUEST`: Deletes PDN session record
-

# User Data Answer (Sh UDA)

The User Data Answer is sent by the HSS to the AS (Application Server) via Sh interface.

## Data Source Mapping



### Key Features:

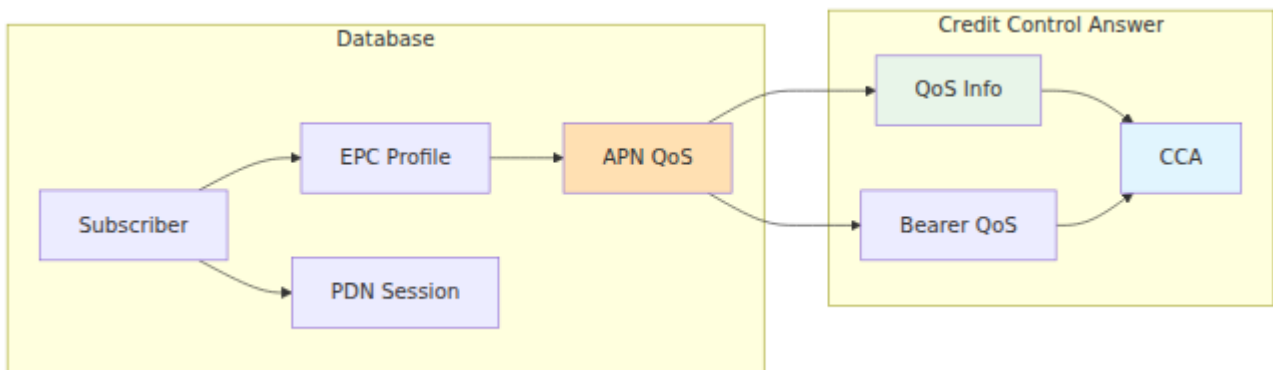
1. **Repository Data:** Can store custom XML in `subscriber_state.sh_repository_data`
2. **Service Indication:** Filters data by requested service (e.g., presence, messaging)
3. **Public Identities:** Returns all IMS public identities for subscriber
4. **Reference vs Transparent:** Supports both reference and transparent data modes

---

## ME Identity Check Answer (S13 ECA)

The ME Identity Check Answer is sent by the EIR function to the MME for IMEI validation.

# Data Source Mapping



## Key Features:

1. **IMEI Regex Matching:** Rules use regular expressions for flexible matching
2. **TAC-based Rules:** Can match Type Allocation Code (first 8 digits)
3. **Default Behavior:** Configurable for unknown IMEIs (accept or reject)
4. **Equipment Status Values:**
  - 0 = WHITELIST (explicitly allowed)
  - 1 = BLACKLIST (stolen/blocked)
  - 2 = GREYLIST (allowed but monitored)
  - 5 = UNKNOWN (no matching rule)

## Use Cases:

- Block stolen devices by exact IMEI
- Block device models by TAC pattern
- Whitelist approved devices only
- Track grey market devices

---

# Common Response Elements

All Diameter responses share these common AVPs:



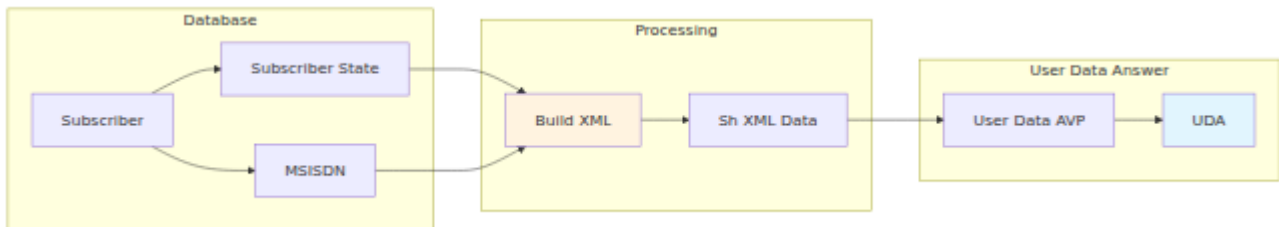
### Configuration Example:

```
config :diameter_ex,
  diameter_host: "hss",
  diameter_realm: "example.com",
  diameter_service_name: "OmniHSS"
```

---

# Data Flow Summary

## Request Processing Pipeline



## Implementation Notes

### Protocol Handlers

The system implements handlers for the following Diameter protocols:

- **S6a** - LTE/MME interface for authentication and location updates
- **Cx** - IMS/CSCF interface for IMS registration and server assignment
- **Sh** - IMS/AS interface for subscriber data retrieval
- **Gx** - PCRF interface for policy and charging control
- **Rx** - IMS/AF interface for media authorization
- **S13** - EIR interface for IMEI validation
- **SWx** - WiFi/IMS interface for non-3GPP access authentication

### Data Models

The database schema includes the following core entities:

- **Subscriber** - Core subscriber record with IMSI
- **Key Set** - Cryptographic keys for authentication
- **EPC Profile** - LTE service configuration
- **APN Profile** - Access point configuration
- **IMS Profile** - IMS service configuration with IFC templates

- **Roaming Profile** - Roaming rules and restrictions
  - **Subscriber State** - Dynamic session and state tracking
  - **PDN Session** - Active bearer session tracking
  - **Static IP** - Static IP address assignments
  - **EIR Rule** - IMEI validation rules
- 

[← Back to Operations Guide](#) | [API Reference](#) → | [Protocol Flows](#) →

# OmniHSS Metrics and Monitoring Guide

[← Back to Operations Guide](#)

---

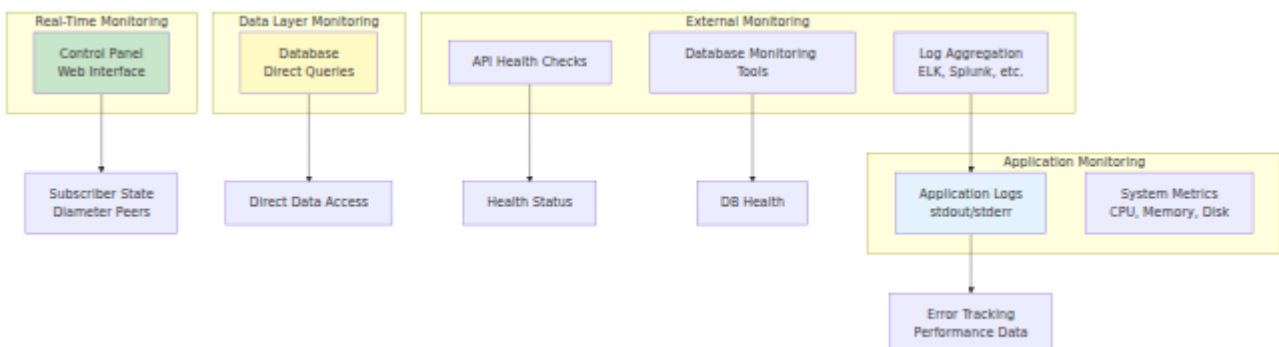
## Table of Contents

- [Monitoring Overview](#)
  - [Control Panel Monitoring](#)
  - [Database Monitoring](#)
  - [Log Monitoring](#)
  - [External Monitoring Integration](#)
  - [Key Performance Indicators](#)
  - [Alerting Strategies](#)
- 

## Monitoring Overview

OmniHSS provides several mechanisms for monitoring system health, performance, and subscriber activity. Operations staff should utilize a combination of these tools for comprehensive visibility.

## Monitoring Layers



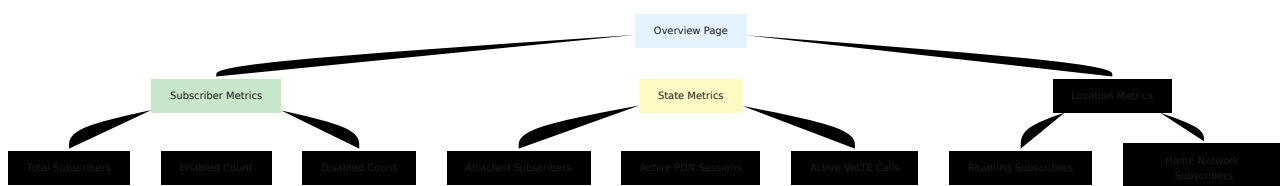
# Control Panel Monitoring

The Control Panel provides the primary real-time monitoring interface.

## Overview Page Monitoring

**URL:** `https://[hostname]:7443/overview`

### Key Metrics Available



### Monitored Subscriber States

State	Indicator	What It Means
<b>Idle</b>	No location info	Subscriber powered off or out of coverage
<b>Attached</b>	MME present	Subscriber registered to network
<b>PDN Active</b>	PDN session count > 0	Active data connection
<b>IMS Registered</b>	S-CSCF assigned	Voice services ready
<b>In Call</b>	Active call count > 0	VoLTE call in progress

### Extracting Metrics from Overview

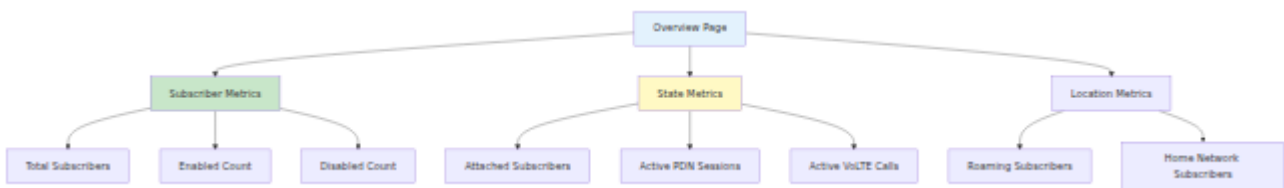
While the Control Panel doesn't export metrics directly, you can:

1. **Count visible rows** for total subscribers
2. **Scan for green checkmarks** to count enabled subscribers
3. **Review expanded details** for state information
4. **Note last seen timestamps** for responsiveness

## Diameter Page Monitoring

**URL:** `https://[hostname]:7443/diameter`

### Key Metrics



### Critical Peer Monitoring

Identify critical peers and monitor their status:

Peer Type	Criticality	Impact if Down
<b>MME</b>	High	No new LTE attachments
<b>P-GW</b>	High	No data sessions
<b>S-CSCF</b>	High	No IMS registrations
<b>P-CSCF</b>	High	No VoLTE calls
<b>I-CSCF</b>	Medium	IMS routing issues
<b>AS</b>	Low-Medium	Specific service unavailable

## Application Page Monitoring

**URL:** `https://[hostname]:7443/application`

## Key Metrics

Metric	Description	Normal Range	Action Threshold
<b>Process Count</b>	Active Erlang processes	Varies by load	> 90% of limit
<b>Memory Usage</b>	Total memory consumed	< 80%	> 90%
<b>Uptime</b>	Time since last restart	N/A	Track for stability

---

# Database Monitoring

## Direct Database Queries

Connect to SQL Database to extract detailed metrics:

### Subscriber Counts

Query the database to retrieve:

- Total count of all subscribers
- Count of enabled subscribers
- Count of IMS-enabled subscribers

### Session Statistics

Query the database to retrieve:

- Count of active PDN sessions
- Count of active VoLTE calls
- Breakdown of PDN sessions by APN profile

## Location Statistics

Query the database to retrieve:

- Subscriber count grouped by visited network (MCC-MNC combination)
- Count of subscribers currently roaming (not on home PLMN 001-001)
- Distribution of subscribers across different visited networks

## Recent Activity

Query the database to retrieve:

- Count of subscribers seen in the last hour
- Distribution of subscribers by serving MME
- Timestamp analysis of last subscriber activity

## Database Health Monitoring

Monitor database health by querying:

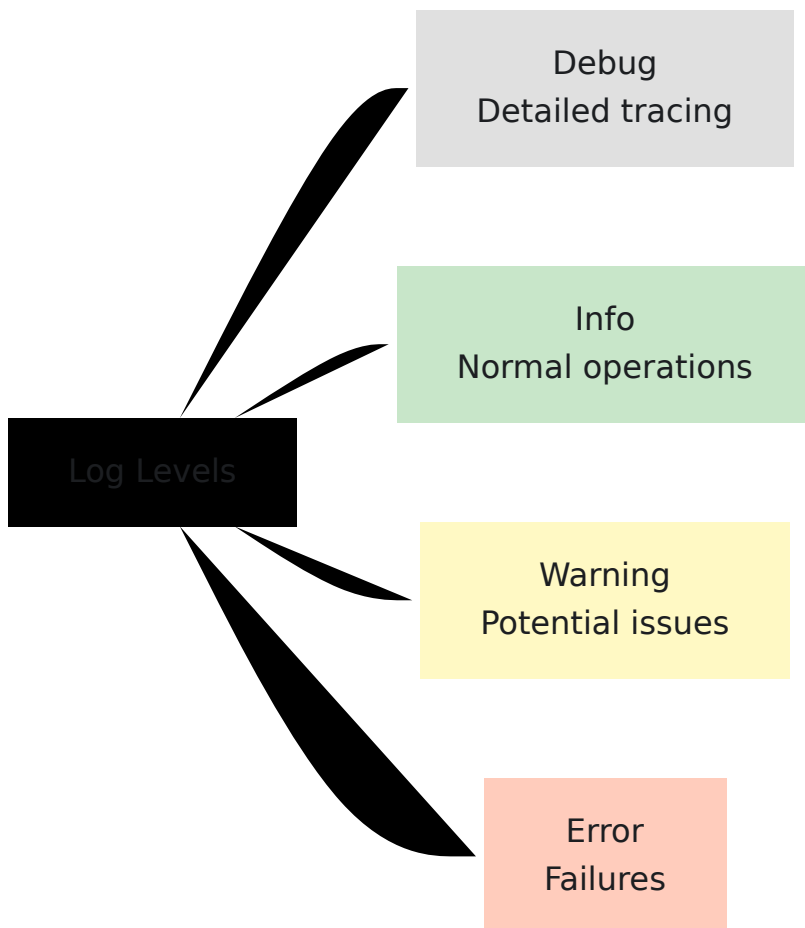
- Total database size and growth trends
  - Individual table sizes and row counts
  - Current database connection count
  - Query performance and resource usage
- 

# Log Monitoring

## Log Output

OmniHSS outputs logs to **stdout/stderr**, which should be captured by your process manager.

## Log Levels



## Key Log Patterns to Monitor

### Diameter Peer Events:

```
[info] Diameter peer connected: mme01.epc.example.com  
[warn] Diameter peer disconnected: pgw01.epc.example.com  
[error] Diameter peer connection failed: timeout
```

### Database Events:

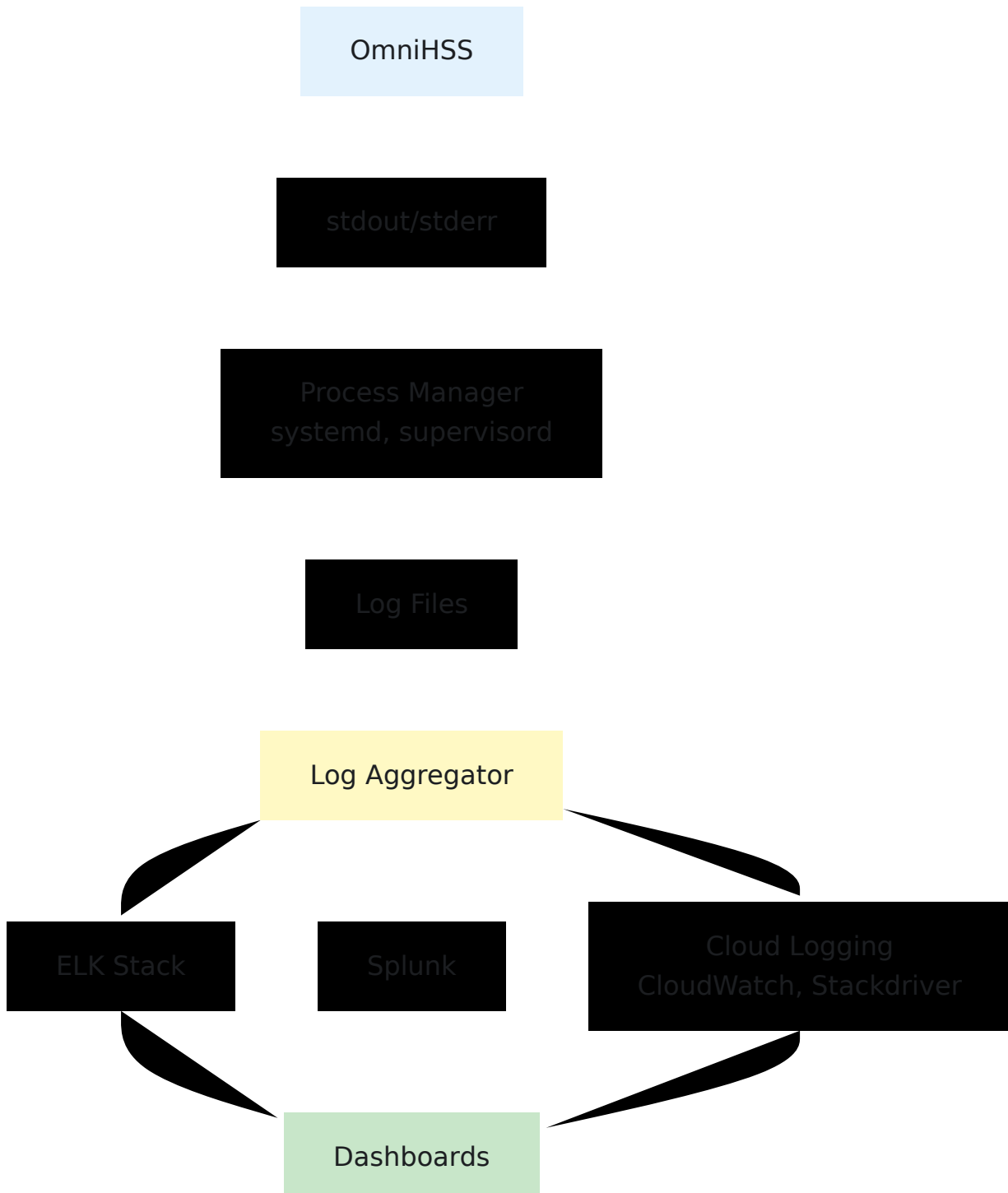
```
[info] Database connection established  
[error] Database connection lost: timeout  
[error] Database query failed: deadlock detected
```

### Authentication Events:

```
[info] Authentication successful: IMSI 001001123456789
[warn] Authentication failed: IMSI 001001123456789, invalid vector
[error] Roaming denied: IMSI 001001123456789, MCC 310 MNC 410
```

## Log Aggregation

For production deployments, implement log aggregation:



---

# External Monitoring Integration

## Health Check Endpoint

**API Health Check:** GET /api/status

```
curl -k https://hss.example.com:8443/api/status
```

### Expected Response:

```
{"status": "ok"}
```

**HTTP Status:** 200 OK

## Monitoring Tool Integration

### Nagios/Icinga Example

```
#!/bin/bash
# check_omnihss.sh

API_URL="https://hss.example.com:8443/api/status"

response=$(curl -k -s -o /dev/null -w "%{http_code}" "$API_URL" --max-time 5)

if [ "$response" = "200" ]; then
    echo "OK - OmniHSS API responding"
    exit 0
else
    echo "CRITICAL - OmniHSS API not responding (HTTP $response)"
    exit 2
fi
```

### Prometheus Integration

Custom exporters can be created to export OmniHSS metrics to Prometheus by querying the API and database.

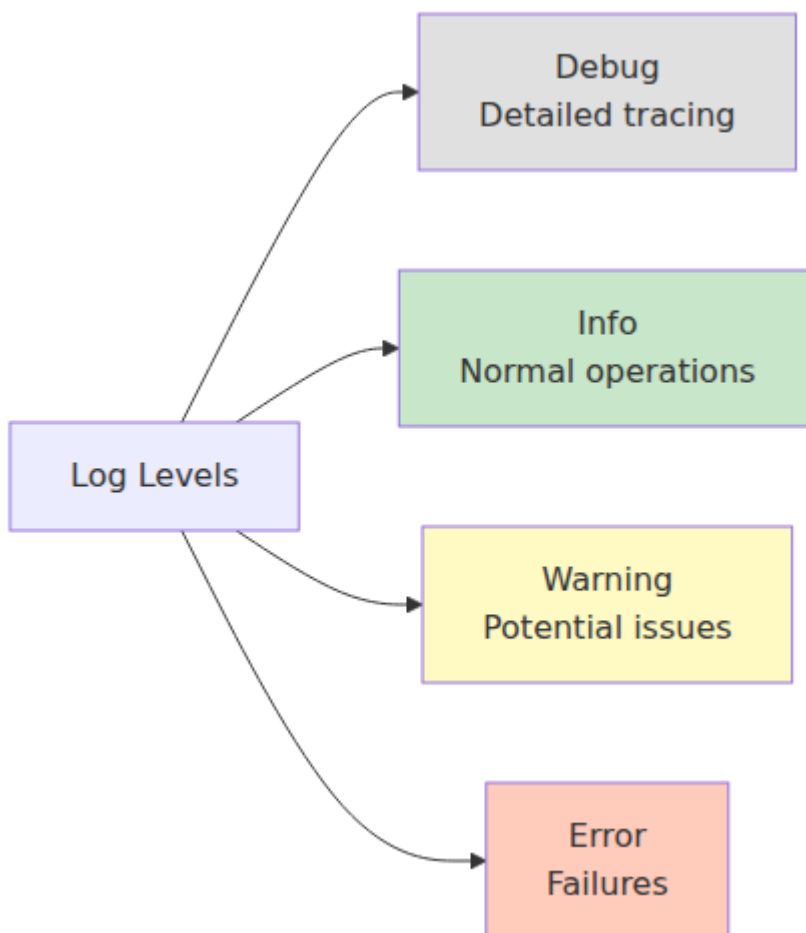
## SNMP Integration

For SNMP-based monitoring, custom SNMP extension scripts can query the database or API for metrics and return values via SNMP OIDs.

---

# Key Performance Indicators

## Operational KPIs



## Recommended KPI Thresholds

KPI	Target	Warning	Critical
<b>System Uptime</b>	99.99%	< 99.95%	< 99.9%
<b>Diameter Peer Uptime</b>	99.9%	< 99.5%	< 99%
<b>Authentication Success Rate</b>	> 99%	< 99%	< 95%
<b>Diameter Response Time</b>	< 100ms	> 200ms	> 500ms
<b>Database Query Time</b>	< 50ms	> 100ms	> 500ms
<b>Error Rate</b>	< 0.1%	> 0.5%	> 1%

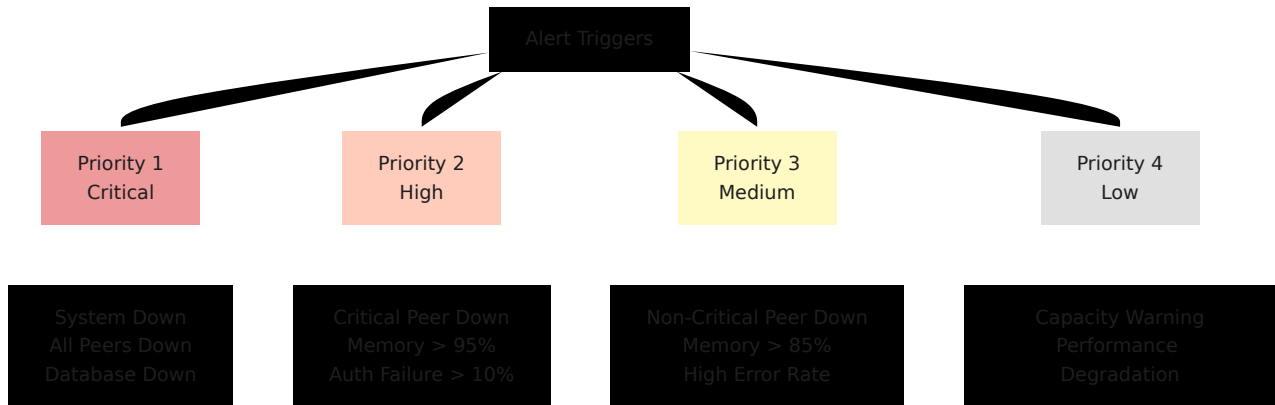
## Capacity KPIs

Metric	Monitor	Plan Action At
<b>Total Subscribers</b>	Current count	80% of expected capacity
<b>Concurrent PDN Sessions</b>	Active sessions	70% of expected maximum
<b>Database Size</b>	MB used	80% of allocated storage
<b>Database Connections</b>	Active connections	80% of pool size

---

# Alerting Strategies

## Alert Priorities



## Alert Definitions

### Critical Alerts (P1)

#### System Unavailable:

- API health check fails
- Control Panel inaccessible
- Database connection fails
- Action: Immediate investigation and escalation

#### All Diameter Peers Disconnected:

- Zero connected peers
- Action: Check network, restart if necessary

#### Database Down:

- Cannot connect to SQL Database
- Action: Investigate database server, restart if necessary

### High Priority Alerts (P2)

#### Critical Diameter Peer Down:

- Primary MME disconnected
- Primary P-GW disconnected
- Primary S-CSCF disconnected
- Action: Investigate peer connectivity within 15 minutes

### **High Memory Usage:**

- Memory > 95%
- Action: Investigate memory leak, plan restart

### **High Authentication Failure Rate:**

- | 10% of auth requests fail
- Action: Check subscriber provisioning, investigate cause

### **Medium Priority Alerts (P3)**

#### **Non-Critical Peer Down:**

- Secondary peer disconnected
- Application Server disconnected
- Action: Investigate within 1 hour

#### **Elevated Memory Usage:**

- Memory > 85%
- Action: Monitor trend, plan capacity upgrade

#### **Elevated Error Rate:**

- Error rate > 1%
- Action: Review logs, identify root cause

### **Low Priority Alerts (P4)**

#### **Capacity Warning:**

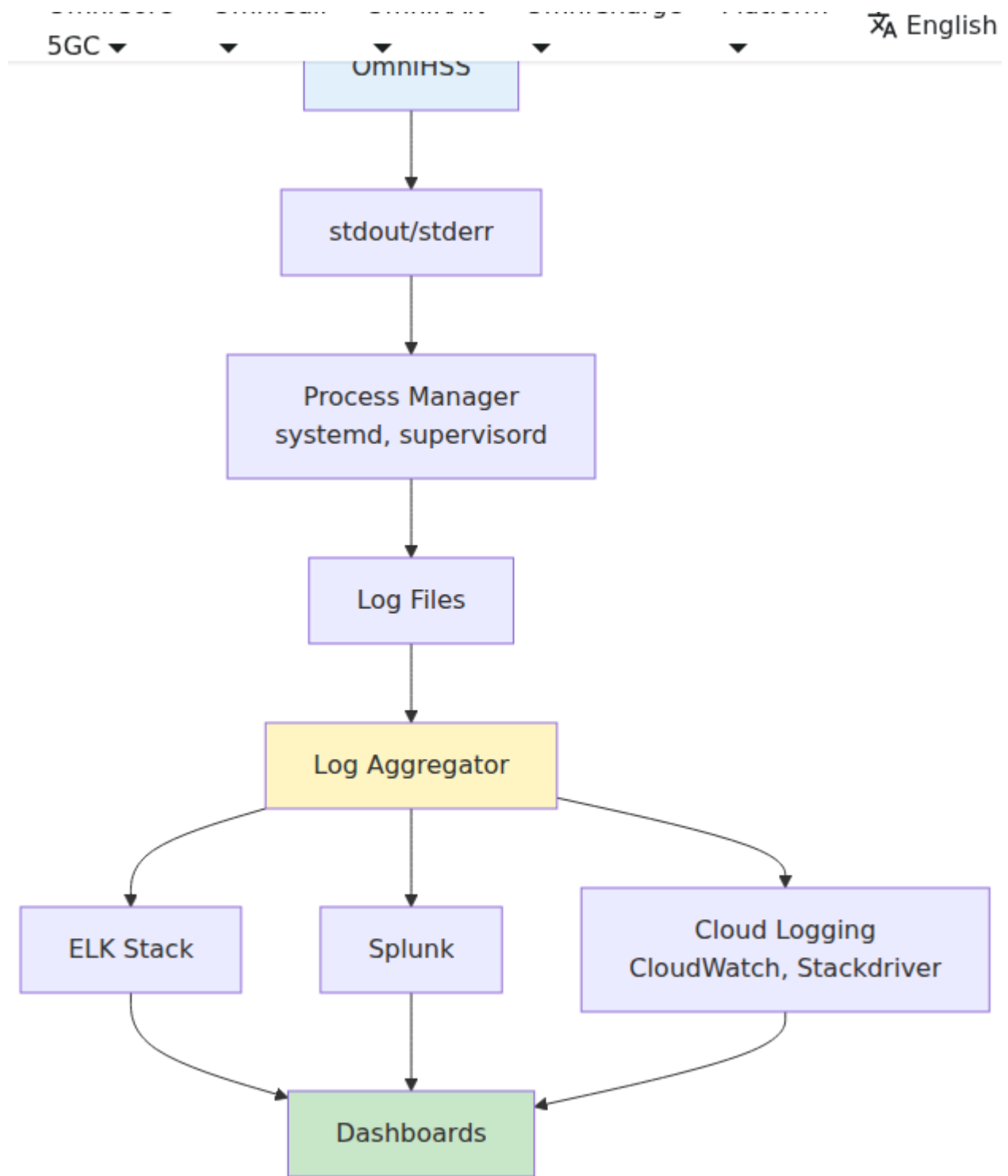
- Subscribers > 80% of capacity
- Database > 80% of allocated storage

- Action: Plan capacity expansion

**Performance Degradation:**

- Response times elevated but acceptable
- Action: Monitor and optimize queries

# Alert Notification Channels



# Monitoring Checklist

## Daily Checks

- Review Control Panel Overview - subscriber counts normal
- Review Diameter page - all critical peers connected
- Review Application page - memory and processes within limits
- Check for error logs - no critical errors in last 24 hours
- Verify backup completed successfully

## Weekly Checks

- Review capacity trends - subscriber growth
- Review performance trends - response times
- Review database size - growth rate acceptable
- Review error rates - identify patterns
- Test alert notifications - ensure working

## Monthly Checks

- Capacity planning review - project 6 months ahead
- Performance optimization review - identify slow queries
- Security review - certificate expiration, vulnerabilities
- Documentation review - update runbooks
- Disaster recovery test - verify backups restore correctly

---

[← Back to Operations Guide](#) | [Next: Multi-Features →](#)

# OmniHSS Multi-MSISDN and Multi-IMSI Features

[← Back to Operations Guide](#)

---

## Table of Contents

- [Overview](#)
  - [Multi-MSISDN: Multiple Phone Numbers](#)
  - [Multi-IMSI SIM: Multiple Network Identities](#)
  - [Combined Scenarios](#)
  - [Configuration Examples](#)
  - [Operational Procedures](#)
- 

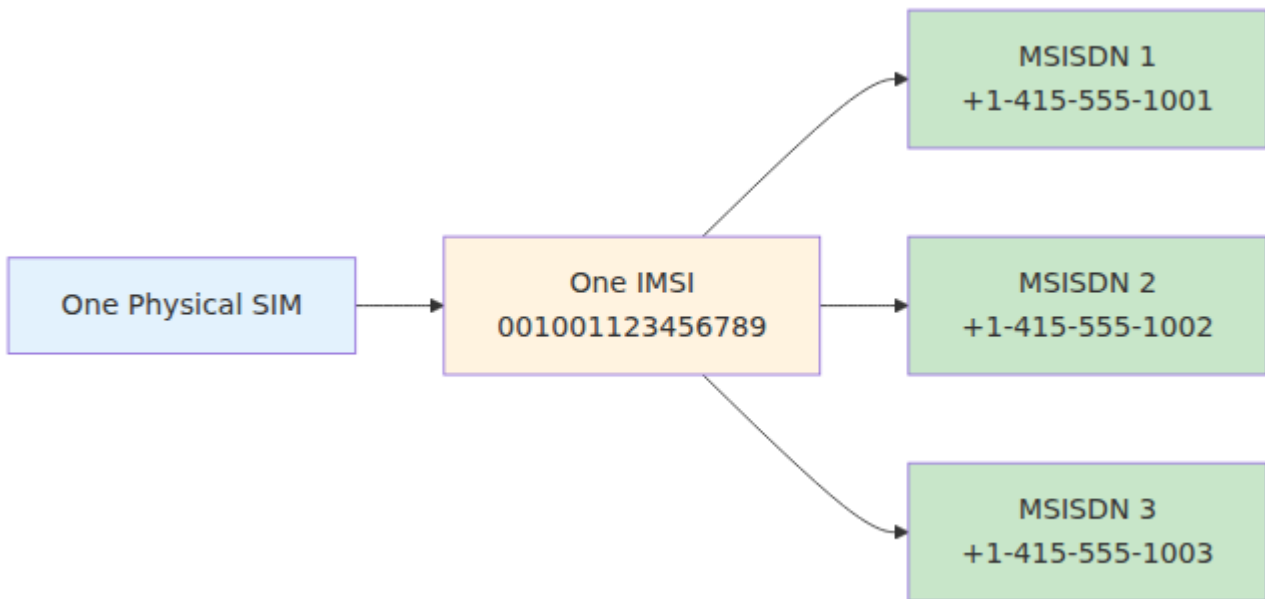
## Overview

OmniHSS supports advanced provisioning capabilities that enable flexible service configurations:

### Multi-MSISDN Support

#### One IMSI → Multiple Phone Numbers

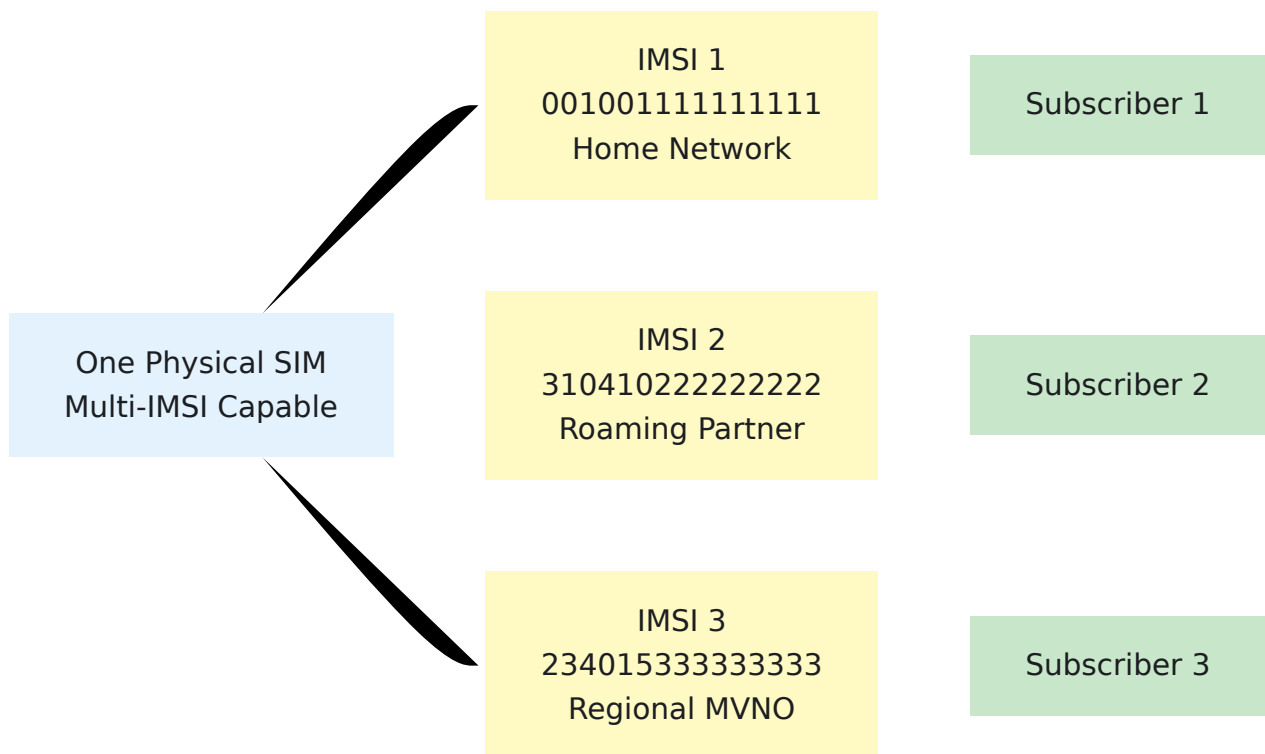
A single subscriber (identified by one IMSI) can have multiple MSISDNs (phone numbers) assigned. All numbers ring on the same device and share the same service profiles.



## Multi-IMSI SIM Support

### One SIM → Multiple IMSIs

A single physical SIM card can contain multiple IMSIs, allowing the device to attach to different networks using different network identities. This is useful for international roaming and MVNO scenarios.



# Multi-MSISDN: Multiple Phone Numbers

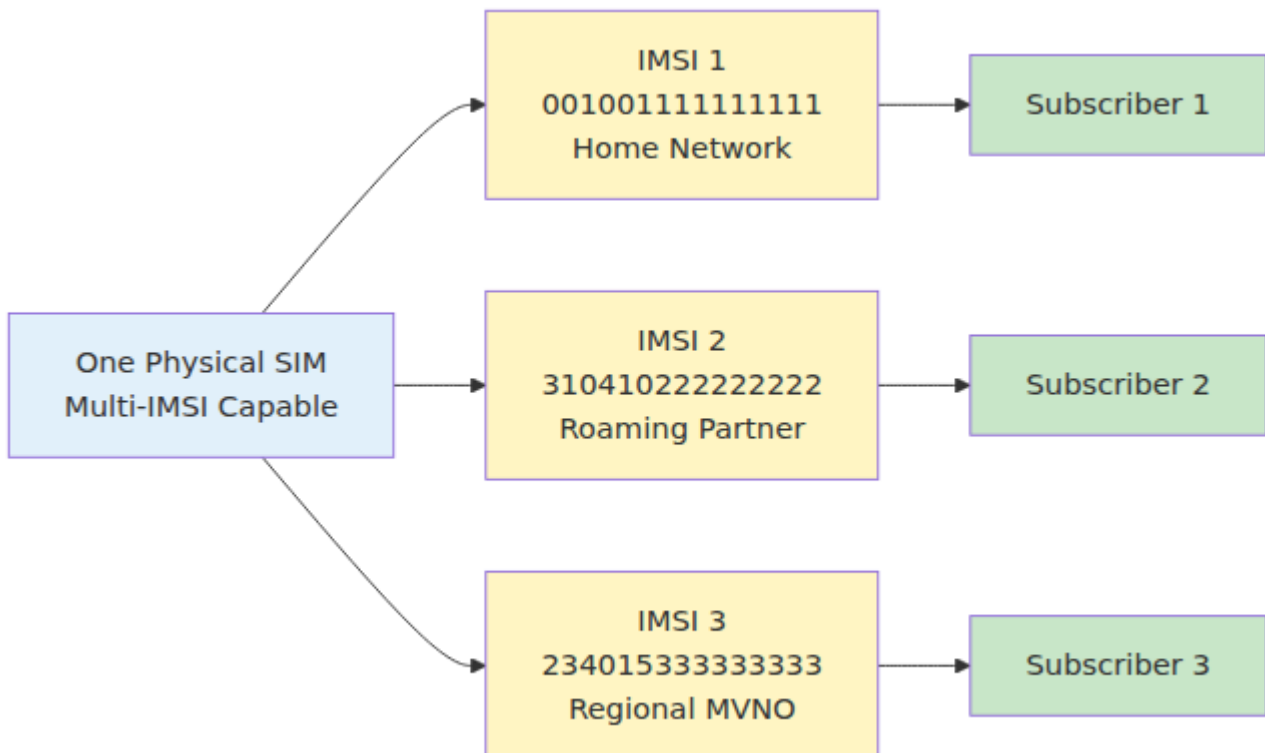
## How It Works

One subscriber record in the HSS has multiple MSISDNs linked through a join table. When the subscriber registers to IMS, all MSISDNs are included in the IMS profile, allowing inbound calls to any number to reach the device.

## Key Characteristics

- **One IMSI** - Subscriber has a single IMSI tied to their SIM card
- **Multiple MSISDNs** - Subscriber can have multiple phone numbers
- **IMS Integration** - All MSISDNs are registered in IMS
- **Shared Service** - All numbers share the same service profiles (EPC, IMS, Roaming)

## Data Model

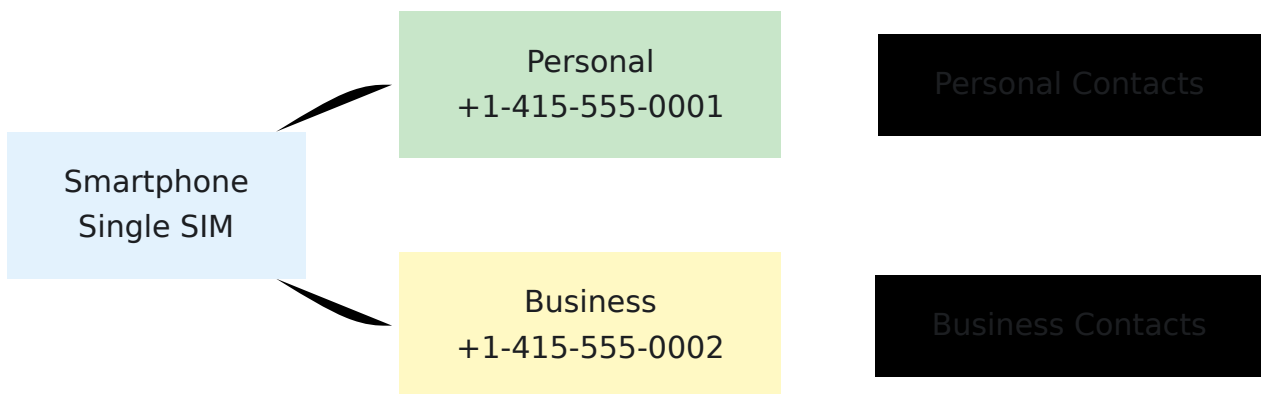


**Important:** One MSISDN can only be assigned to ONE subscriber at a time. However, one subscriber can have MANY MSISDNs.

## Use Cases

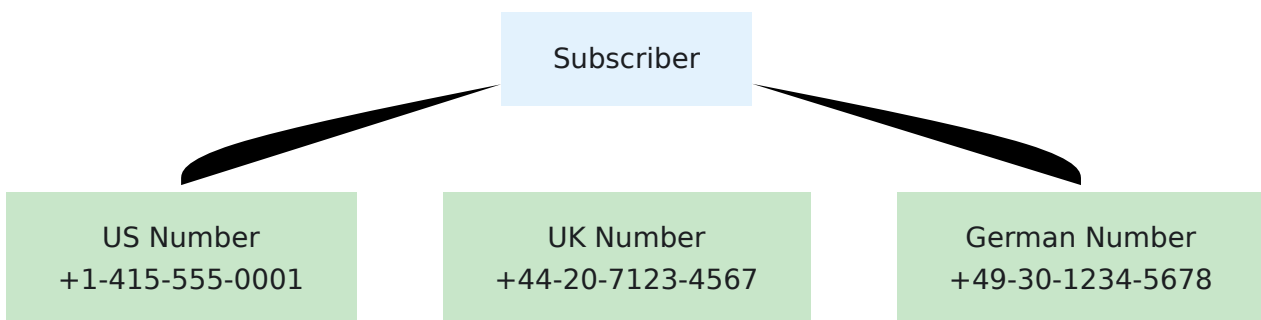
### 1. Business and Personal Lines

A subscriber has both business and personal phone numbers on the same device:



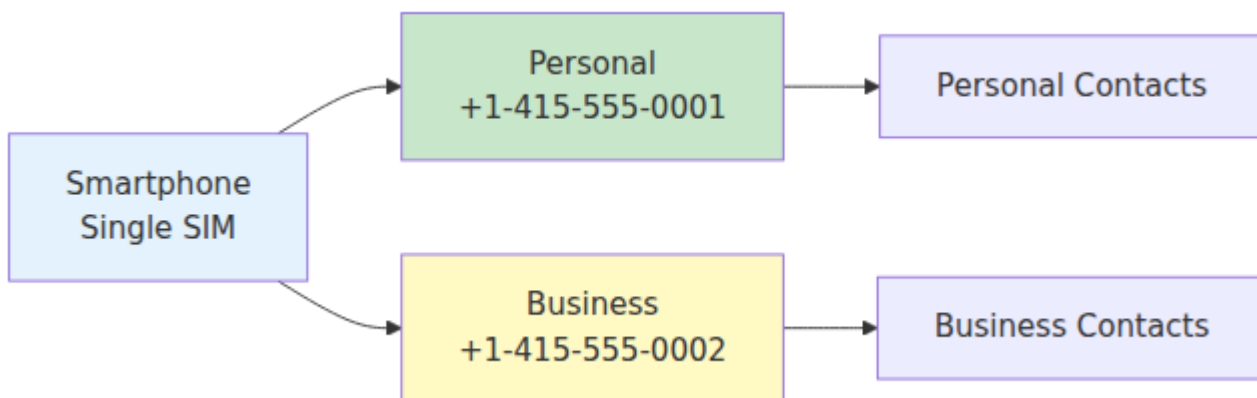
### 2. International Numbers

A subscriber who travels frequently has numbers in multiple countries:



### 3. Family Plans

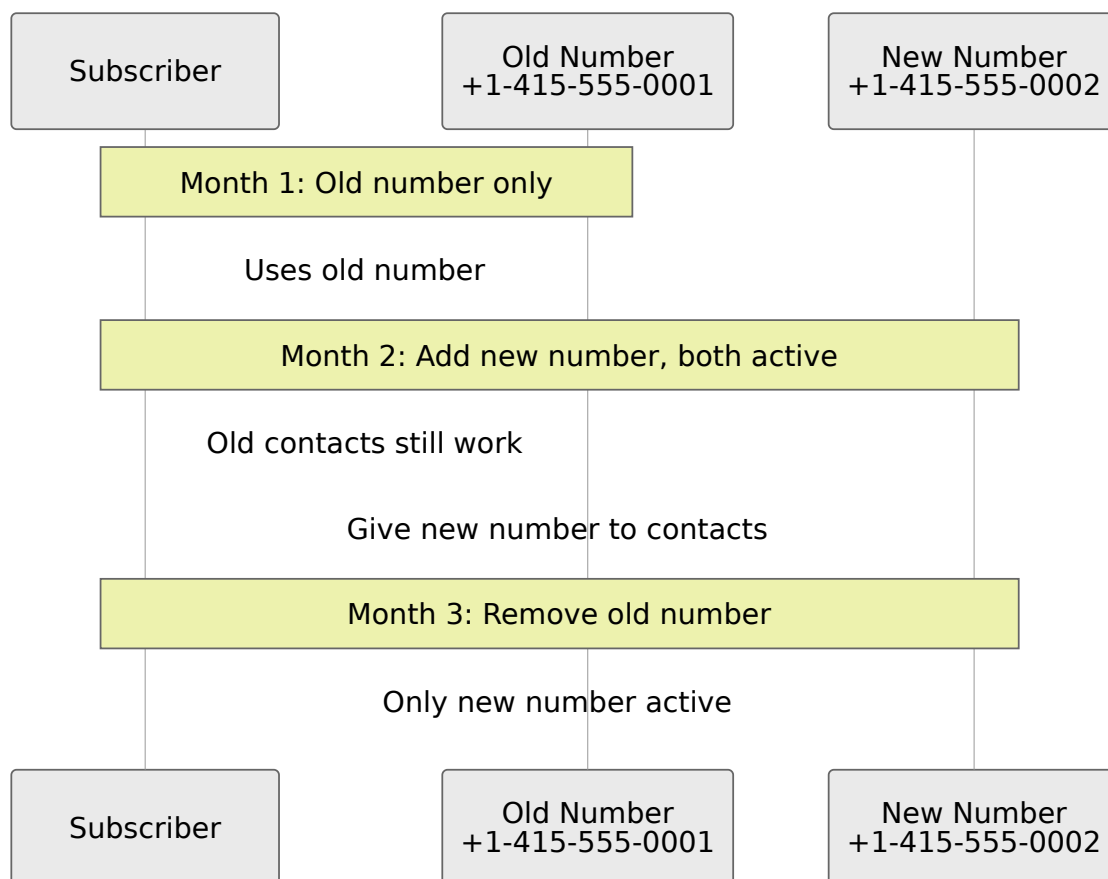
One parent manages multiple family member numbers:



**Note:** In OmniHSS, this would require multiple subscribers (one per SIM/IMSI), each potentially having multiple MSISDNs.

#### 4. Legacy Line Porting

When a subscriber changes numbers but wants to keep the old number active during transition:



## Configuration

### Creating MSISDNs

MSISDNs must be created before assigning to subscribers.

```
# Create first MSISDN
curl -k -X POST https://hss.example.com:8443/api/msisdn \
  -H "Content-Type: application/json" \
  -d '{"msisdn": {"msisdn": "14155551001"}}'

# Create second MSISDN
curl -k -X POST https://hss.example.com:8443/api/msisdn \
  -H "Content-Type: application/json" \
  -d '{"msisdn": {"msisdn": "14155551002"}}'
```

## Assigning MSISDNs to Subscribers

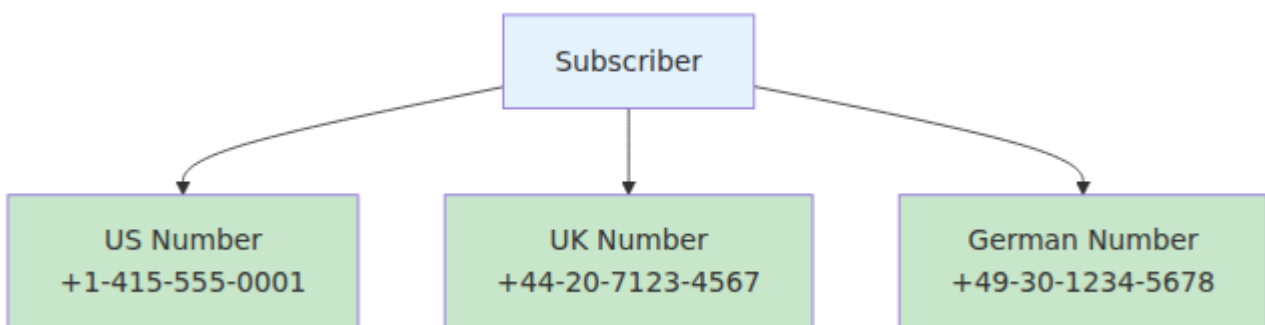
The assignment is done through the join table in the database.

### Database Method:

1. Query the database to get the subscriber ID for the target IMSI
2. Query the database to get the MSISDN IDs for the phone numbers
3. Insert records into the join table linking subscriber\_id to each msisdn\_id

This creates the many-to-many relationship between the subscriber and their phone numbers.

### Provisioning Workflow



### Verifying Assignment

Query the database to retrieve the subscriber along with all linked MSISDNs by:

- Joining the subscriber table with the join table

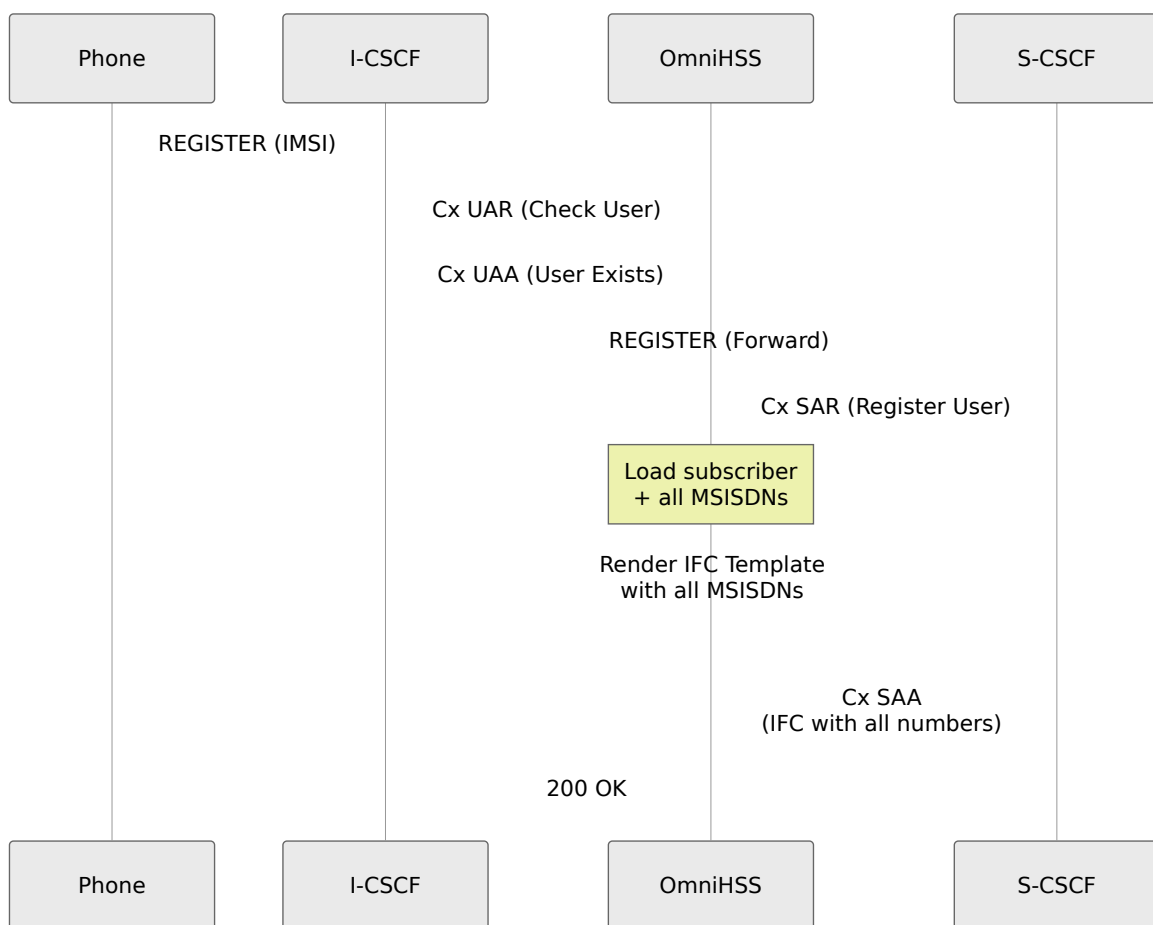
- Joining the join table with the msisdn table
- Grouping results by subscriber to see all phone numbers together

This will show the subscriber ID, IMSI, and a list of all assigned MSISDNs.

## IMS Integration

### IMS Registration

When a subscriber registers to IMS, **all assigned MSISDNs are included** in the IMS profile sent to the S-CSCF.



### IFC Template Rendering

The IMS IFC template can reference all MSISDNs using the `{{msisdns}}` variable.

### Example IFC Template:

```

<ServiceProfile>
  <PublicIdentity>
    <Identity>sip:
{{imsi}}@ims.mnc{{mnc}}.mcc{{mcc}}.3gppnetwork.org</Identity>
  </PublicIdentity>
  <!-- Repeat for each MSISDN -->
  <PublicIdentity>
    <Identity>sip:+14155551001@ims.example.com</Identity>
  </PublicIdentity>
  <PublicIdentity>
    <Identity>tel:+14155551001</Identity>
  </PublicIdentity>
  <PublicIdentity>
    <Identity>sip:+14155551002@ims.example.com</Identity>
  </PublicIdentity>
  <PublicIdentity>
    <Identity>tel:+14155551002</Identity>
  </PublicIdentity>
  <!-- ... -->
</ServiceProfile>

```

### Template Variable:

- `{{msisdns}}` - List of all MSISDNs assigned to subscriber

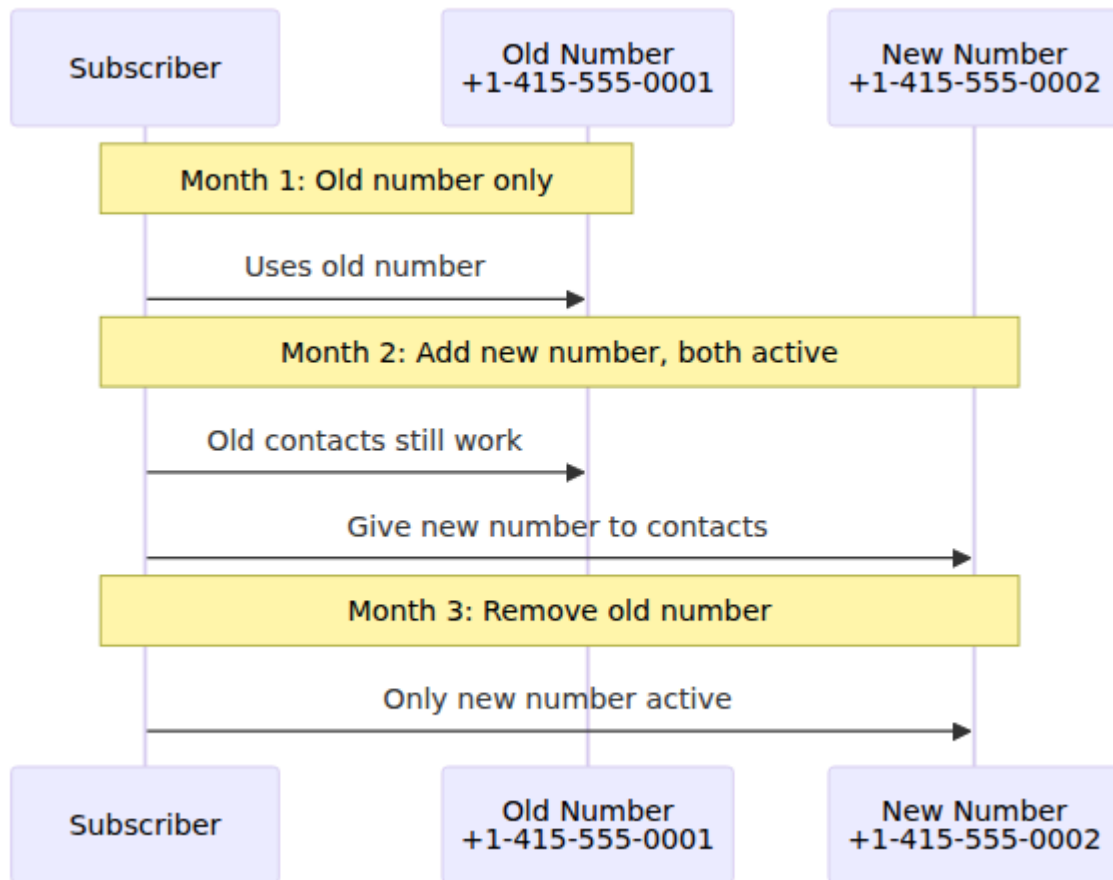
### Public Identities

Each MSISDN typically results in two IMS public identities:



### Inbound Call Routing

When someone calls one of the subscriber's numbers, the IMS network routes to the correct SIP URI:



## Outbound Call Presentation

The phone can choose which number to present as caller ID for outbound calls.

### SIP INVITE Example:

```
INVITE sip:+15105551234@ims.example.com SIP/2.0
From: "+14155551002" <sip:+14155551002@ims.example.com>;tag=123
To: <sip:+15105551234@ims.example.com>
P-Asserted-Identity: <sip:+14155551002@ims.example.com>
```

The `From` and `P-Asserted-Identity` headers indicate which of the subscriber's numbers is being used.

# Troubleshooting Multi-MSISDN

## Issue: MSISDN Not Appearing in IMS Registration

### Symptoms:

- S-CSCF shows only one public identity
- Calls to second number fail

### Troubleshooting Steps:

#### 1. Verify MSISDN Assignment in Database:

- Query the database to retrieve all MSISDNs linked to the subscriber's IMSI
- Check the join table to ensure the relationships exist

#### 2. Check IMS Profile Template:

- Verify template includes `{{msisdns}}` variable
- Confirm template syntax is valid XML

#### 3. Review HSS Logs:

- Look for IMS registration (Cx SAR) messages
- Verify all MSISDNs are included in response

#### 4. Test IMS Registration:

- Trigger re-registration on phone
- Check S-CSCF logs for public identities registered

## Issue: Cannot Assign MSISDN to Subscriber

### Symptoms:

- Database insert fails
- Error: "Duplicate entry" or "Foreign key constraint"

### Possible Causes:

## 1. MSISDN Already Assigned:

- Query the database to check if the MSISDN is already linked to another subscriber
- **Solution:** Remove the existing assignment first, then create the new assignment

## 2. MSISDN Doesn't Exist:

- Query the database to verify the MSISDN record exists
- **Solution:** Create the MSISDN record first via API or database insert

## Issue: Calls to One Number Work, Other Doesn't

### Symptoms:

- Calls to primary number work
- Calls to secondary number fail or route incorrectly

### Troubleshooting Steps:

#### 1. Verify Both Numbers in IMS Registration:

- Check S-CSCF registered public identities
- Confirm both SIP URIs present

#### 2. Check IMS Routing Rules:

- Verify IFC template routing rules apply to all identities
- Check if specific number needs special routing

#### 3. Test Both Numbers:

```
# Test from SIP client
sip:+14155551001@ims.example.com # Should work
sip:+14155551002@ims.example.com # Should also work
```

## Issue: API Lookup by MSISDN Returns Wrong Subscriber

### Symptoms:

- API query `/api/subscriber/msisdn/:msisdn` returns unexpected subscriber

### **Verification:**

Query the database to find which subscriber the MSISDN is assigned to. This should return exactly one subscriber. If it returns multiple or the wrong subscriber, the join table has incorrect data that needs to be corrected.

## **Best Practices**

### **Provisioning Order**

1. Create all MSISDNs first
2. Create subscriber
3. Assign MSISDNs to subscriber
4. Verify assignment before activation

### **MSISDN Management**

- **Document primary vs secondary** numbers in subscriber custom\_attributes
- **Port numbers sequentially** when porting to avoid service disruption
- **Test all numbers** after provisioning before giving to customer

### **IMS Configuration**

- Ensure IFC template handles multiple public identities correctly
- Test inbound routing to all numbers
- Verify caller ID presentation for outbound calls

### **Migration**

When migrating from single to multi-MSISDN:

Subscriber has 1 MSISDN

Add second MSISDN

Test both numbers

Both work?

Yes

No

Activate for customer

Debug issue

Complete

---

# Multi-IMSI SIM: Multiple Network Identities

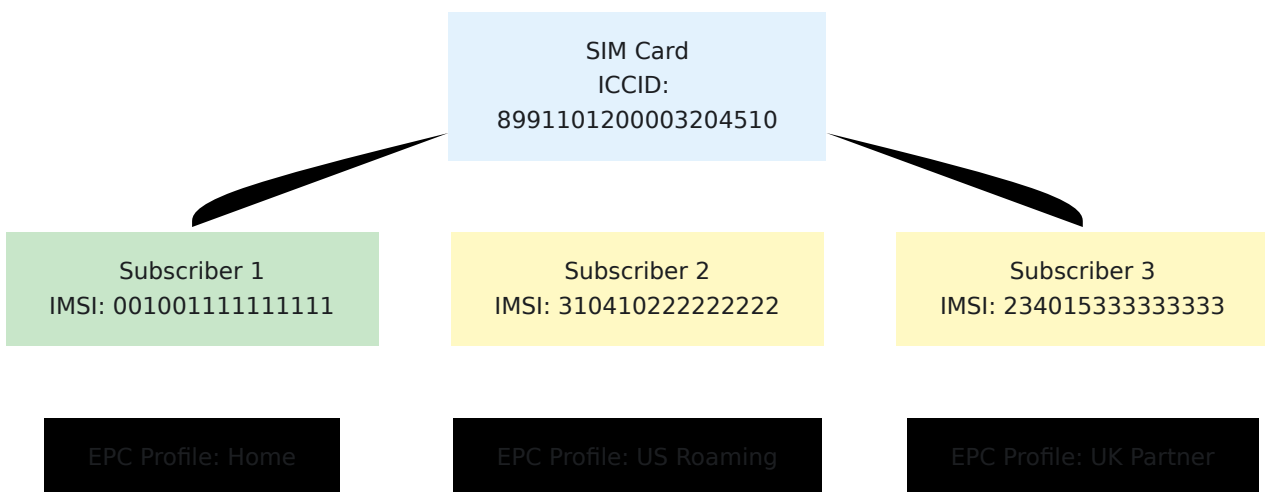
## How It Works

A multi-IMSI SIM contains multiple complete subscriber profiles, each with its own IMSI, keys, and credentials. The device can switch between IMSIs to attach to different networks, often automatically based on location or network availability.

**Important:** Only **one IMSI can be active at any given time**. When a device switches to a different IMSI on the same SIM card, the HSS will automatically deregister the previously active IMSI.

## OmniHSS Implementation

In OmniHSS, each IMSI on a multi-IMSI SIM is provisioned as a **separate subscriber record**, but all reference the **same SIM card**:



## Use Cases

### 1. International Roaming Optimization

- Home IMSI: 001-001 (home network rates)
- US Roaming IMSI: 310-410 (local US rates)

- EU Roaming IMSI: 234-015 (local EU rates)
- Device switches IMSI based on location

## **2. MVNO Service**

- Primary IMSI: MVNO network (reseller)
- Fallback IMSI: Host network (parent operator)
- Automatic failover if MVNO coverage unavailable

## **3. IoT/M2M Multi-Network**

- IMSI 1: Primary carrier
- IMSI 2: Backup carrier for redundancy
- IMSI 3: Emergency/low-cost fallback
- Critical devices maintain connectivity

## **4. Regulatory Compliance**

- Different IMSIs for different regulatory zones
- Comply with local data residency requirements
- Use local network identity per jurisdiction

# **Multi-IMSI Features**

## **Independent Authentication**

- Each IMSI has its own Ki, OPC, and key set
- Separate authentication vectors per IMSI
- Different security credentials per network

## **Separate Service Profiles**

- Different EPC profiles (bandwidth, APNs)
- Different IMS profiles (voice services)
- Different roaming rules per IMSI

## **Shared Physical Identity**

- All IMSIs reference same SIM (via sim\_id)
- Same ICCID across all subscriber records
- Logical grouping via SIM card

### **Network Selection**

- Device or SIM card decides which IMSI to use
- Based on available networks, location, policy
- HSS authenticates whichever IMSI device presents

# Configuration

```
# 1. Create SIM card (multi-IMSI capable)
SIM_ID=$(curl -k -X POST https://hss.example.com:8443/api/sim \
  -d '{"sim": {"iccid": "8991101200003204510", "is_esim": false}}' \
  \
  | jq -r '.data.id')

# 2. Create key set for IMSI 1 (home network)
KEYSET1=$(curl -k -X POST https://hss.example.com:8443/api/key_set \
  \
  -d '{"key_set": {"ki": "0123456789ABCDEF...", "opc": \
  "FEDCBA9876..."}}' \
  | jq -r '.data.id')

# 3. Create subscriber 1 (home IMSI)
curl -k -X POST https://hss.example.com:8443/api/subscriber \
  -d '{"subscriber": {
    "imsi": "\0010011111111111",
    "sim_id": $SIM_ID,
    "key_set_id": $KEYSET1,
    "epc_profile_id": 1
  }}'

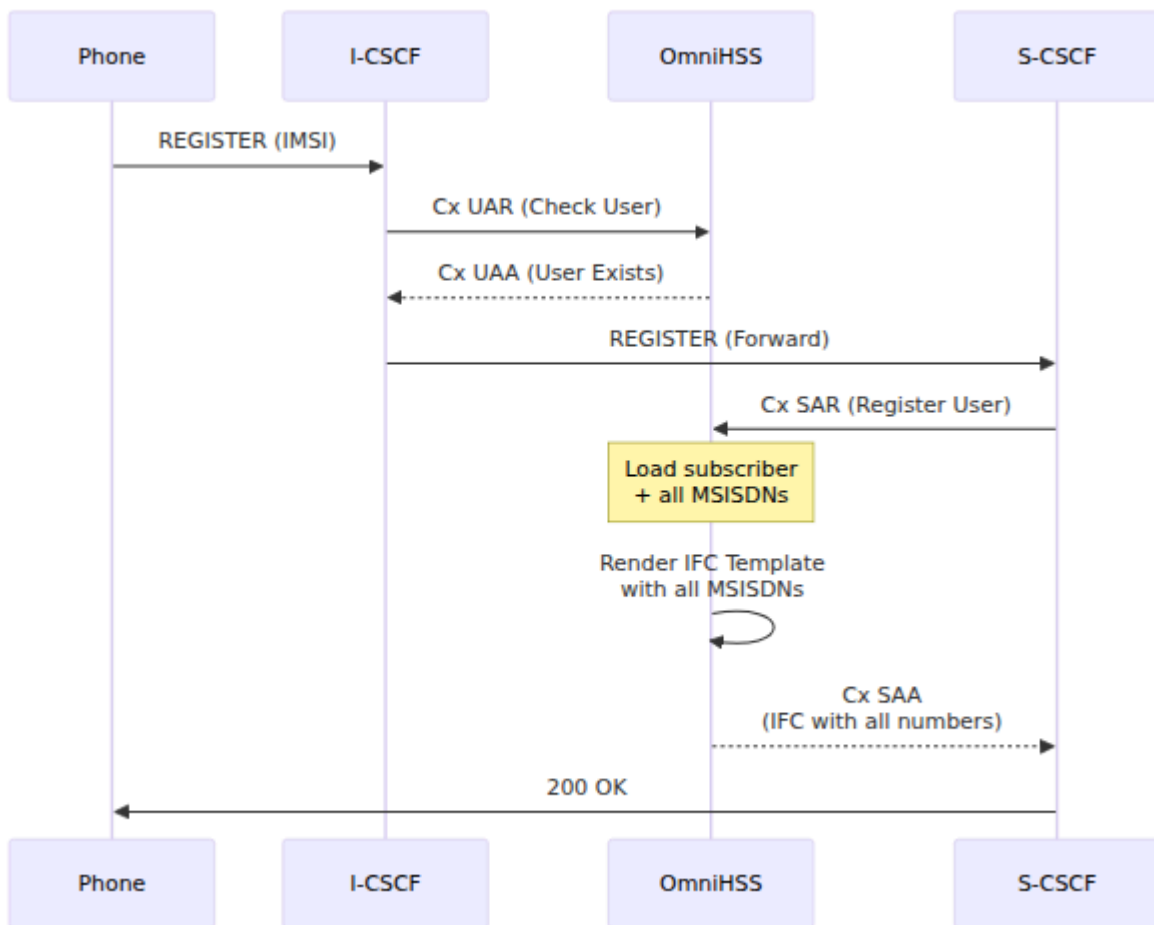
# 4. Create key set for IMSI 2 (roaming partner)
KEYSET2=$(curl -k -X POST https://hss.example.com:8443/api/key_set \
  \
  -d '{"key_set": {"ki": "1111111111111111...", "opc": \
  "2222222222..."}}' \
  | jq -r '.data.id')

# 5. Create subscriber 2 (roaming IMSI)
curl -k -X POST https://hss.example.com:8443/api/subscriber \
  -d '{"subscriber": {
    "imsi": "\3104102222222222",
    "sim_id": $SIM_ID,
    "key_set_id": $KEYSET2,
    "epc_profile_id": 2
  }}'

# 6. Repeat for additional IMSIs on the SIM...
```

# Authentication Flow

When a multi-IMSI device attaches:



The HSS doesn't need to know it's a multi-IMSI SIM—it just authenticates whatever IMSI the device presents.

## IMSI Switching and Automatic Deregistration

When a multi-IMSI SIM switches from one IMSI to another, only one IMSI can be registered at a time on the network. OmniHSS automatically handles this by sending a **Cancel Location Request (CLR)** to deregister the previously active IMSI when a new IMSI from the same SIM card registers.

### Single Active IMSI Rule

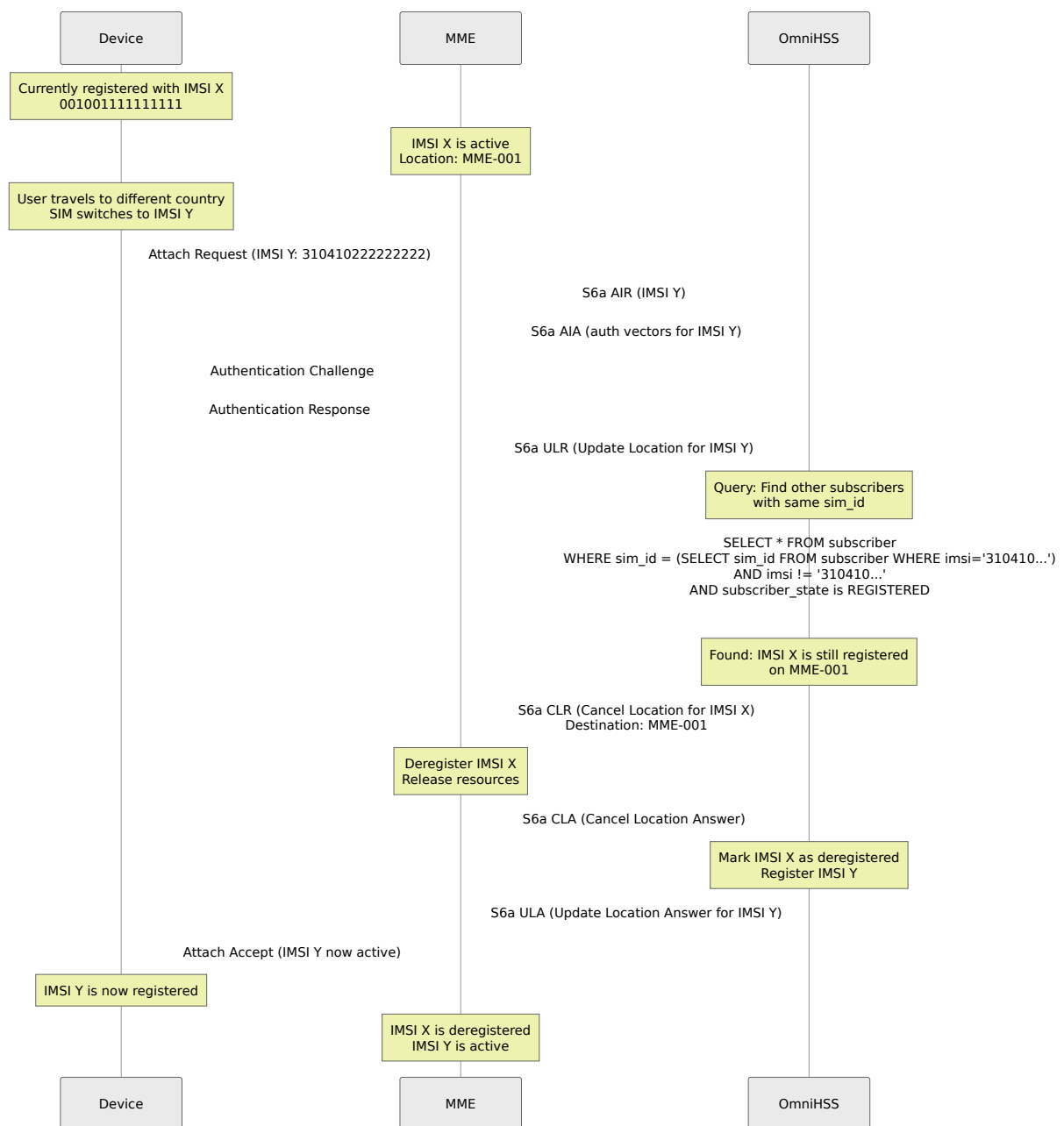
**Key Concept:** Only one subscriber (IMSI) per SIM card can be active at any given time.

- If a subscriber is registered on an MME using **IMSI X**

- And the HSS receives an Update Location Request for **IMSI Y** (on the same SIM as IMSI X)
- The HSS automatically sends a **Cancel Location Request** to deregister **IMSI X**

This ensures clean handoff between IMSIs and prevents conflicts in the network.

### IMSI Switching Flow



### Why This Matters

## **Network Integrity:**

- Prevents duplicate registrations from the same physical SIM
- Ensures network resources are properly released
- Maintains accurate subscriber location data

## **Billing Accuracy:**

- Only one IMSI is charged for network access at a time
- Clear session boundaries between IMSI switches
- Accurate CDR (Call Detail Record) generation

## **Resource Management:**

- MME resources for old IMSI are freed
- PDP contexts and bearers are cleaned up
- Location tracking remains accurate

## **IMSI Switch Triggers**

The device/SIM decides when to switch IMSIs based on:

### **1. Network Availability**

- Home IMSI network not available
- Switch to roaming partner IMSI

### **2. Manual Selection**

- User manually selects network
- SIM switches to corresponding IMSI

### **3. Policy-Based**

- SIM card has internal rules (e.g., prefer local IMSI in certain countries)
- Automatic switching based on MCC/MNC

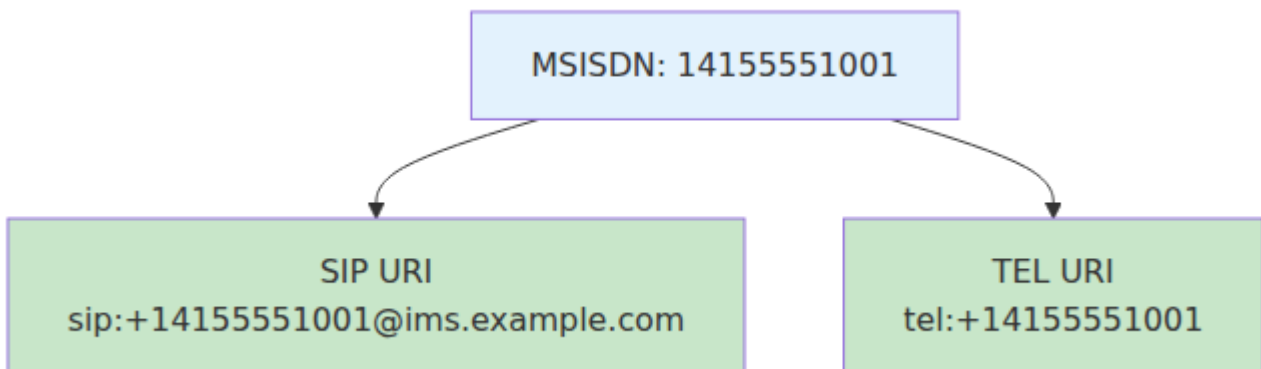
### **4. Cost Optimization**

- Switch to IMSI with lower roaming rates

- Use local IMSI to avoid roaming charges

## IMS Considerations

The same Cancel Location Request behavior applies to IMS registration:



## Operational Impact

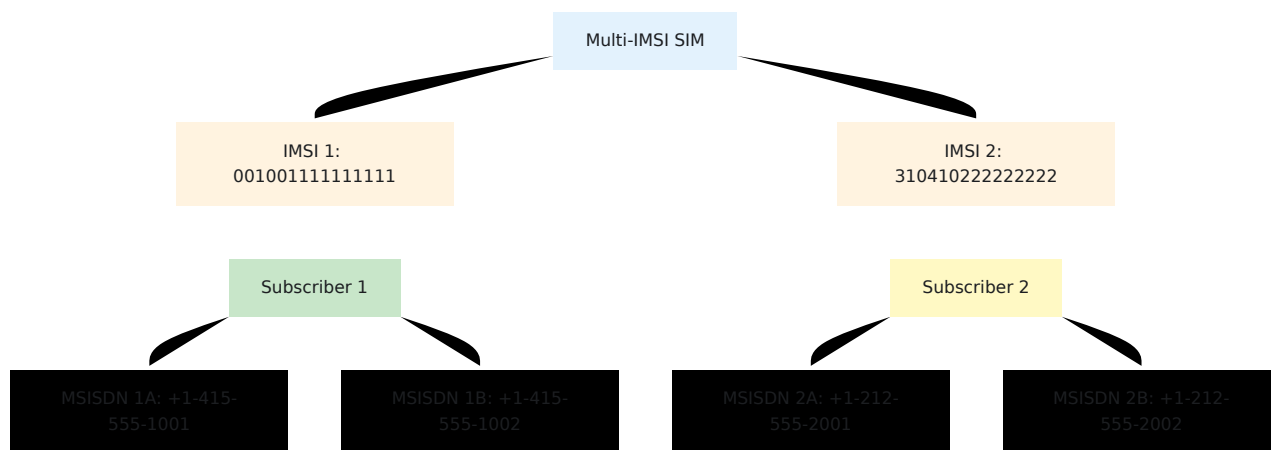
### For Operations Staff:

1. **Subscriber appears offline:** When IMSI switches, the old IMSI will show as "deregistered" in the HSS. This is normal behavior.
  2. **Two subscriber records for one SIM:** Multi-IMSI SIMs will have multiple subscriber records sharing the same `sim_id`. Only one will be in "registered" state at a time.
  3. **Location tracking:** The `subscriber_state` table tracks which MME/SGSN each IMSI is registered with. When IMSI switches, the old location is cleared.
  4. **Troubleshooting:** If a device cannot be reached:
    - Check which IMSI is currently registered
    - Verify the correct IMSI is being used for the current network
    - Confirm only one IMSI per SIM is in registered state
-

# Combined Scenarios

## Multi-IMSI + Multi-MSISDN

You can combine both features: multiple IMSIs on one SIM, each with multiple MSISDNs.



### Example Use Case:

- **Home Network (IMSI 1):**
  - Personal number: +1-415-555-1001
  - Business number: +1-415-555-1002
- **US Roaming Network (IMSI 2):**
  - Personal number: +1-212-555-2001
  - Business number: +1-212-555-2002

When device is in home territory, uses IMSI 1 with its MSISDNs. When roaming in US, switches to IMSI 2 with different MSISDNs optimized for US network.

---

## Operational Procedures

### Managing Multi-MSISDN Subscribers

View all MSISDNs for a subscriber:

```
Query via API: GET /api/subscriber/imsi/:imsi
```

The response includes all linked MSISDNs.

## Troubleshooting Multi-IMSI

### Device not attaching with second IMSI:

1. Verify second subscriber record exists for that IMSI
2. Check key\_set is configured correctly for that IMSI
3. Verify EPC profile is assigned
4. Confirm roaming rules allow attachment

### Device switching IMSIs unexpectedly:

- This is controlled by device/SIM logic, not HSS
- HSS authenticates whatever IMSI is presented
- Check device IMSI selection settings

## Troubleshooting Multi-MSISDN

### Second number not ringing:

1. Verify MSISDN is linked in join table
2. Check IMS profile template includes `{{msisdns}}` variable
3. Confirm IMS registration includes all public identities
4. Review S-CSCF logs for registered identities

### Outbound calls only show one number:

- Device selects which number to present as caller ID
  - This is device configuration, not HSS
  - HSS provides all identities; device chooses
-

# Benefits Summary

## Multi-MSISDN Benefits

✓ One SIM, multiple phone numbers ✓ Separate business and personal lines ✓ International local presence ✓ Simplified device management ✓ All numbers share same data service ✓ Centralized billing per IMSI

## Multi-IMSI SIM Benefits

✓ Optimized roaming costs ✓ Automatic network selection ✓ Redundancy and failover ✓ Local network identity ✓ Regulatory compliance ✓ Service continuity across networks

## Combined Benefits

✓ Maximum flexibility ✓ Different number sets per network ✓ Optimized for each use case ✓ Complex business scenarios ✓ International and local optimization

---

[← Back to Operations Guide](#)

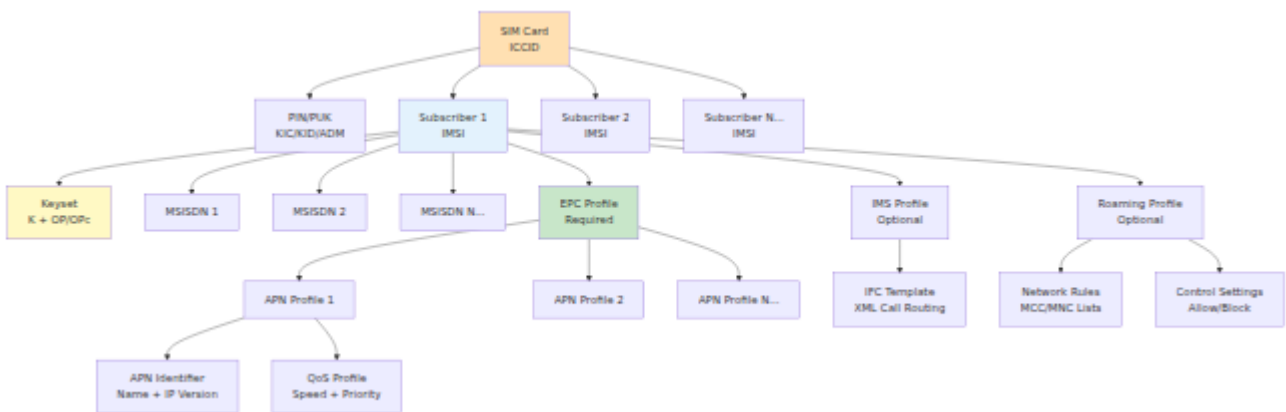
# OmniHSS Profile Management

[← Back to Operations Guide](#)

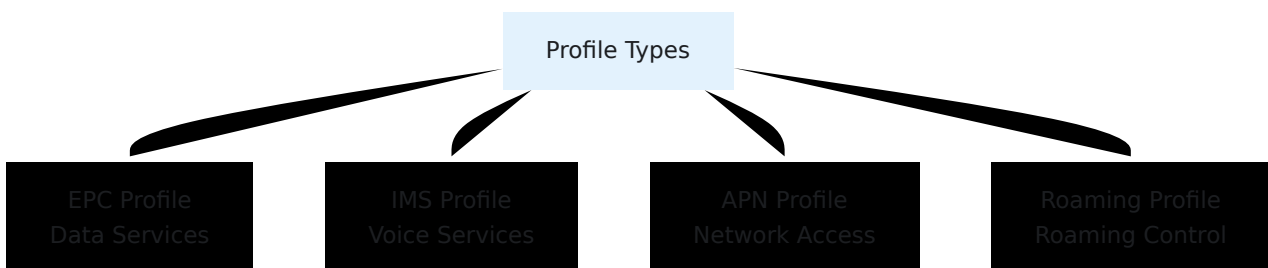
---

## Overview

OmniHSS uses **profiles** to define service characteristics for subscribers. Profiles allow you to create reusable service templates that can be assigned to multiple subscribers, simplifying provisioning and ensuring consistency.



## Profile Types



---

# EPC Profiles

EPC (Evolved Packet Core) Profiles define data service characteristics for LTE subscribers.

## Key Parameters

Parameter	Description	Typical Value
<code>ue_ambr_dl_kbps</code>	Download speed limit	10,000 - 1,000,0 Kbps
<code>ue_ambr_ul_kbps</code>	Upload speed limit	5,000 - 500,000 Kbps
<code>network_access_mode</code>	Service type	"packet_only" or "packet_and_circ"
<code>tracking_area_update_interval_seconds</code>	TAU timer	54 seconds (typi

## Creating EPC Profiles

```
curl -k -X POST https://hss.example.com:8443/api/epc/profile \
-H "Content-Type: application/json" \
-d '{
  "apn_profiles": [],
  "name": "Premium 100Mbps",
  "network_access_mode": "packet_only",
  "tracking_area_update_interval_seconds": 600,
  "ue_ambr_dl_kbps": 100000,
  "ue_ambr_ul_kbps": 50000
}'
```

## Common EPC Profile Templates

**Basic Internet:**

- Download: 10 Mbps (10,000 Kbps)
- Upload: 5 Mbps (5,000 Kbps)

### **Standard:**

- Download: 50 Mbps (50,000 Kbps)
- Upload: 25 Mbps (25,000 Kbps)

### **Premium:**

- Download: 100 Mbps (100,000 Kbps)
- Upload: 50 Mbps (50,000 Kbps)

### **Unlimited:**

- Download: 1 Gbps (1,000,000 Kbps)
  - Upload: 500 Mbps (500,000 Kbps)
- 

## **IMS Profiles**

IMS Profiles define voice service characteristics, primarily through IFC (Initial Filter Criteria) templates.

## **IFC Templates**

IFC templates are XML documents that define call routing rules for the S-CSCF.

### **Template Variables:**

- `{{imsi}}` - Subscriber IMSI
- `{{msisdns}}` - List of phone numbers
- `{{mcc}}` - Home country code
- `{{mnc}}` - Home network code

# Creating IMS Profiles

```
curl -k -X POST https://hss.example.com:8443/api/ims/profile \  
-H "Content-Type: application/json" \  
-d '{  
  "ims_profile": {  
    "name": "Standard VoLTE",  
    "ifc_template": "<InitialFilterCriteria>...  
</InitialFilterCriteria>"  
  }  
'
```

## IFC Template Example

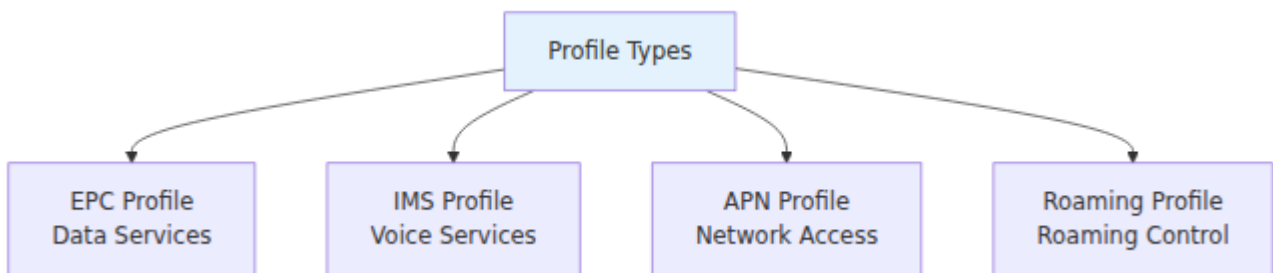
```
<ServiceProfile>  
  <PublicIdentity>  
    <Identity>sip:  
{{imsi}}@ims.mnc{{mnc}}.mcc{{mcc}}.3gppnetwork.org</Identity>  
  </PublicIdentity>  
  <InitialFilterCriteria>  
    <Priority>0</Priority>  
    <TriggerPoint>  
      <ConditionTypeCNF>0</ConditionTypeCNF>  
      <SPT>  
        <ConditionNegated>0</ConditionNegated>  
        <Group>0</Group>  
        <Method>INVITE</Method>  
      </SPT>  
    </TriggerPoint>  
    <ApplicationServer>  
      <ServerName>sip:as.ims.example.com</ServerName>  
      <DefaultHandling>0</DefaultHandling>  
    </ApplicationServer>  
  </InitialFilterCriteria>  
</ServiceProfile>
```

---

# APN Profiles

APN (Access Point Name) Profiles define network access points for data connections.

## APN Components



## APN Identifier

Defines the APN name and IP protocol support.

### Common APNs:

- `internet` - General internet access
- `ims` - IMS/VoLTE signaling
- `mms` - Multimedia messaging
- `vzwadmin` - Carrier-specific

### IP Version Options:

- `"ipv4"`: IPv4 only
- `"ipv6"`: IPv6 only
- `"ipv4v6"`: IPv4v6 (dual stack)
- `"ipv4_or_ipv6"`: IPv4 or IPv6 (network choice)

## APN QoS Profile

Defines quality of service parameters.

### QCI (QoS Class Identifier) Values:

QCI	Type	Use Case	Priority
1	GBR	Conversational voice	Highest
2	GBR	Conversational video	High
4	GBR	Video streaming	High
5	Non-GBR	IMS signaling	Medium
9	Non-GBR	Internet (default)	Lowest

### Session-Level Charging Method (optional):

The APN QoS profile can also pin the default charging method for the IP-CAN session, sent as the top-level `Online/Offline` AVPs in the Gx CCA-I.

Field	Type	Default	Description
<code>online_charging_enabled</code>	boolean	<code>null</code>	When set, forces the session-level <code>Online</code> AVP ( <code>true</code> = enable, <code>false</code> = disable). When <code>null</code> , the PCRF echoes the value the PGW sent in the CCR-I.
<code>offline_charging_enabled</code>	boolean	<code>null</code>	When set, forces the session-level <code>Offline</code> AVP ( <code>true</code> = enable, <code>false</code> = disable). When <code>null</code> , the PCRF echoes the value the PGW sent in the CCR-I.

See [PCRF — Session-Level Online/Offline Charging](#) for the full behaviour and AVP value reference.

# Creating Complete APN Configuration

```
# 1. Create APN Identifier
APN_ID=$(curl -k -X POST
https://hss.example.com:8443/api/apn/identifier \
  -H "Content-Type: application/json" \
  -d '{"apn": "internet", "ip_version": "ipv4v6"}' \
  | jq -r '.response.id')

# 2. Create APN QoS Profile
QOS_ID=$(curl -k -X POST
https://hss.example.com:8443/api/apn/qos_profile \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Best Effort",
    "allocation_retention_priority": 8,
    "apn_ambr_dl_kbps": 50000,
    "apn_ambr_ul_kbps": 25000,
    "pre_emption_capability": false,
    "pre_emption_vulnerability": true,
    "qci": 9
  }' | jq -r '.response.id')

# 3. Create APN Profile
curl -k -X POST https://hss.example.com:8443/api/apn/profile \
  -H "Content-Type: application/json" \
  -d "{
    \"apn_identifier_id\": $APN_ID,
    \"apn_qos_profile_id\": $QOS_ID,
    \"name\": \"Internet APN\"
  }"
```

## Assigning APNs to EPC Profile

APNs are linked to EPC Profiles through the `join_epc_profile_to_apn_profile` table.

Insert records into the join table to link APN profile IDs to the EPC profile ID. Multiple APN profiles can be assigned to one EPC profile.

---

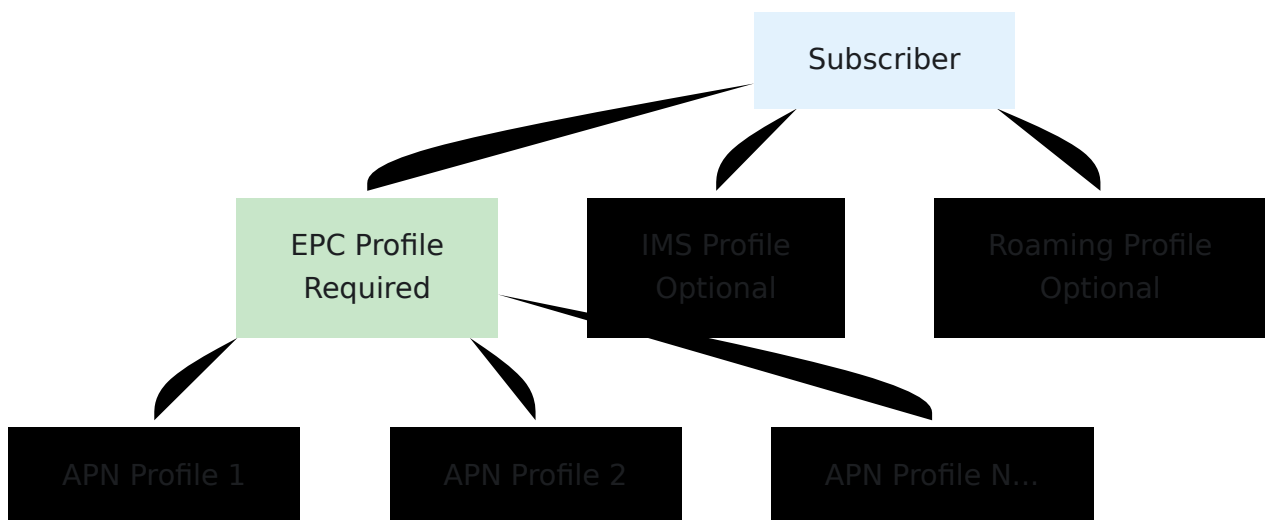
# Roaming Profiles

See detailed documentation in [Roaming Control Guide](#).

---

## Profile Assignment

### Subscriber Profile Relationships



# Assigning Profiles to Subscribers

```
# Assign EPC and IMS profiles during subscriber creation
curl -k -X POST https://hss.example.com:8443/api/subscriber \
  -H "Content-Type: application/json" \
  -d '{
    "subscriber": {
      "imsi": "001001123456789",
      "key_set_id": 1,
      "epc_profile_id": 1,
      "ims_profile_id": 1,
      "roaming_profile_id": 1
    }
  }'
```

```
# Update subscriber profile
curl -k -X PUT https://hss.example.com:8443/api/subscriber/1 \
  -H "Content-Type: application/json" \
  -d '{
    "subscriber": {
      "epc_profile_id": 2
    }
  }'
```

---

## Profile Management Best Practices

### Design Principles

1. **Create Standard Profiles** - Define common service tiers (Basic, Standard, Premium)
2. **Reuse Profiles** - Assign same profile to multiple subscribers
3. **Document Changes** - Track profile modifications
4. **Test Before Production** - Verify profile works with test subscriber first

# Profile Naming Convention

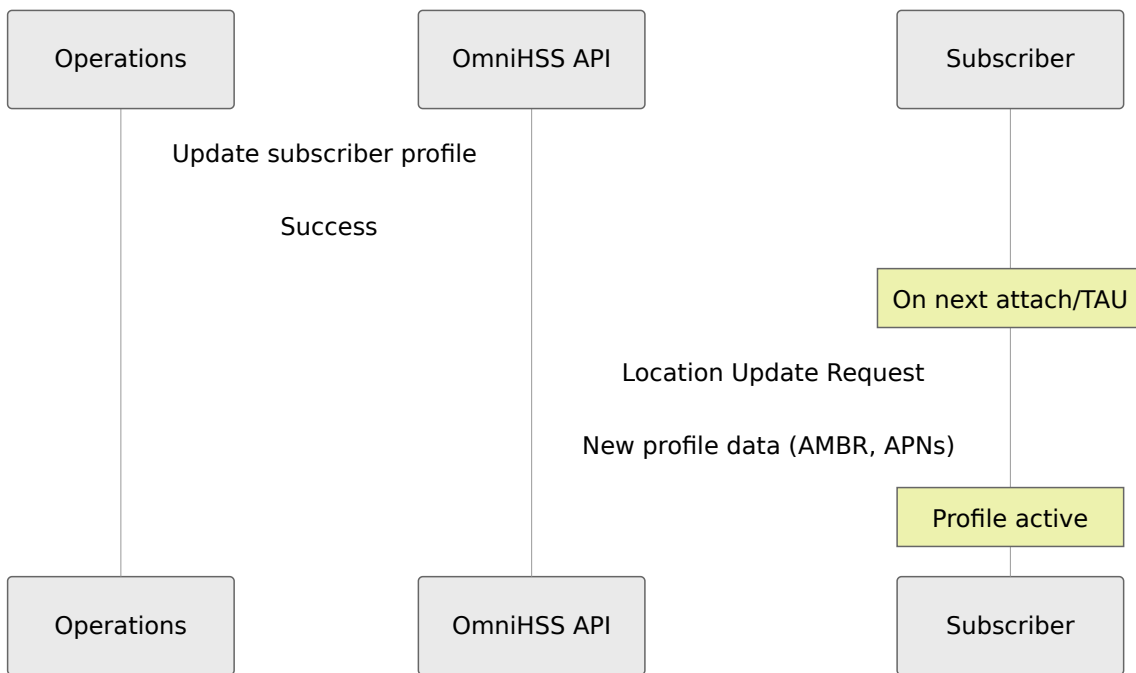
[Service Tier]-[Speed]-[Features]

Examples:

- "Basic-10Mbps-Internet"
- "Premium-100Mbps-VoLTE"
- "Enterprise-1Gbps-MultiAPN"

# Profile Migration

When changing a subscriber's profile:



**Important:** Profile changes take effect on the next:

- Tracking Area Update (TAU)
- Attach
- IMS Registration (for IMS profile changes)

# Troubleshooting Profile Issues

**Subscriber not getting expected speed:**

1. Check assigned EPC profile AMBR values
2. Check APN QoS profile AMBR values
3. Verify MME/P-GW enforcing QoS correctly
4. Check for network congestion

**IMS registration fails:**

1. Verify IMS profile assigned
2. Check IFC template XML validity
3. Review S-CSCF logs for IFC processing errors
4. Confirm S-CSCF selection configuration

**APN not available:**

1. Verify APN profile linked to EPC profile
2. Check APN identifier matches network request
3. Review PDN connectivity request from UE

---

[← Back to Operations Guide](#) | [Next: Roaming Control](#) →

# OmniHSS Protocol Flows

[← Back to Operations Guide](#)

---

## Overview

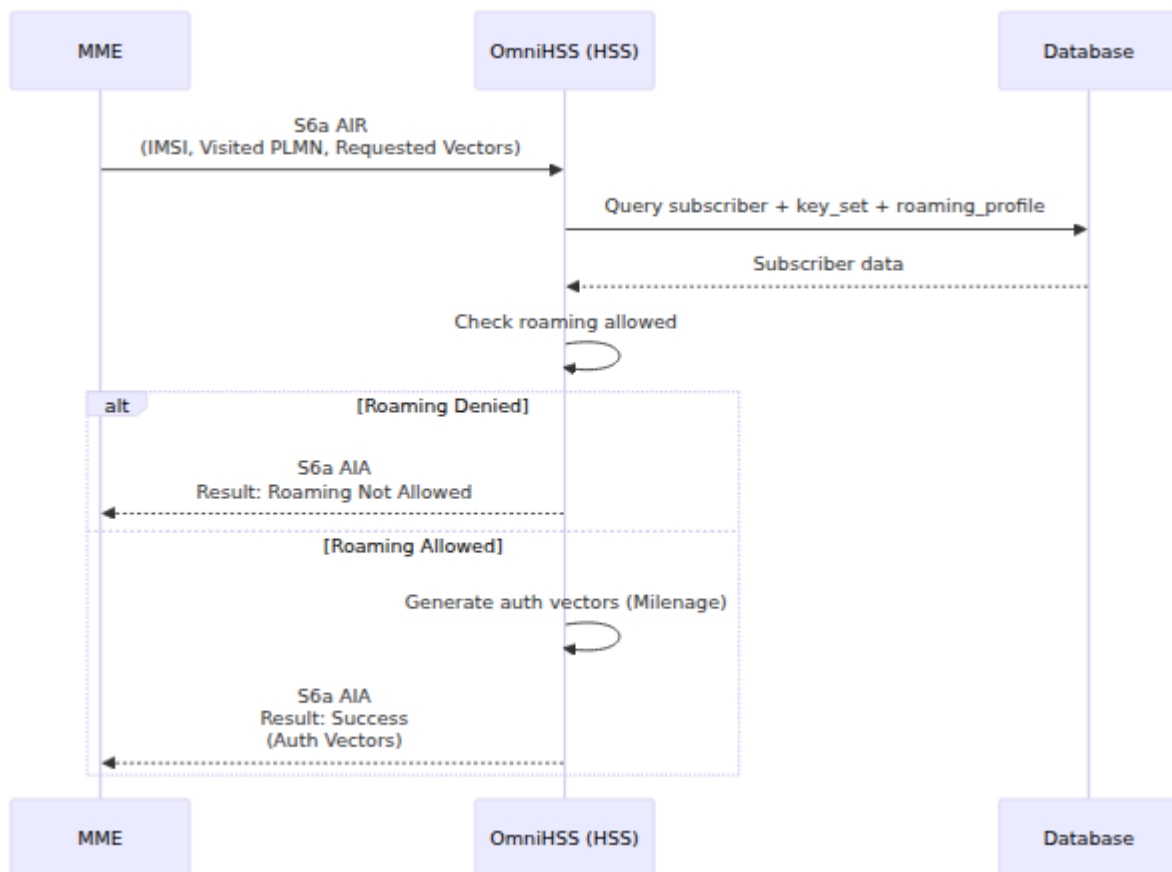
This document details the Diameter protocol message flows supported by OmniHSS. Understanding these flows is essential for troubleshooting and operations.

---

## S6a Interface (LTE/EPC)

### Authentication Information Request (AIR/AIA)

MME requests authentication vectors for subscriber.

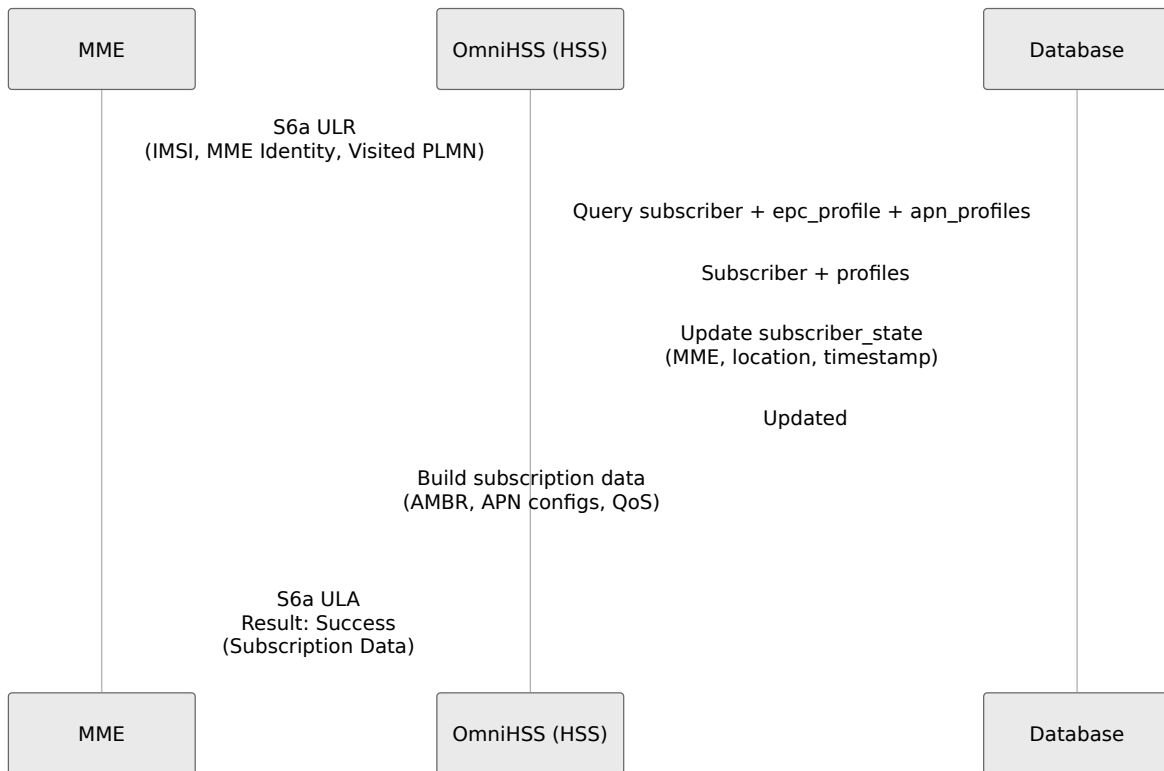


### Key AVPs:

- Request: User-Name (IMSI), Visited-PLMN-Id, Number of Requested Vectors
- Response: Authentication-Info (RAND, AUTN, XRES, KASME)

## Update Location Request (ULR/ULA)

MME notifies HSS of subscriber location and retrieves subscription data.

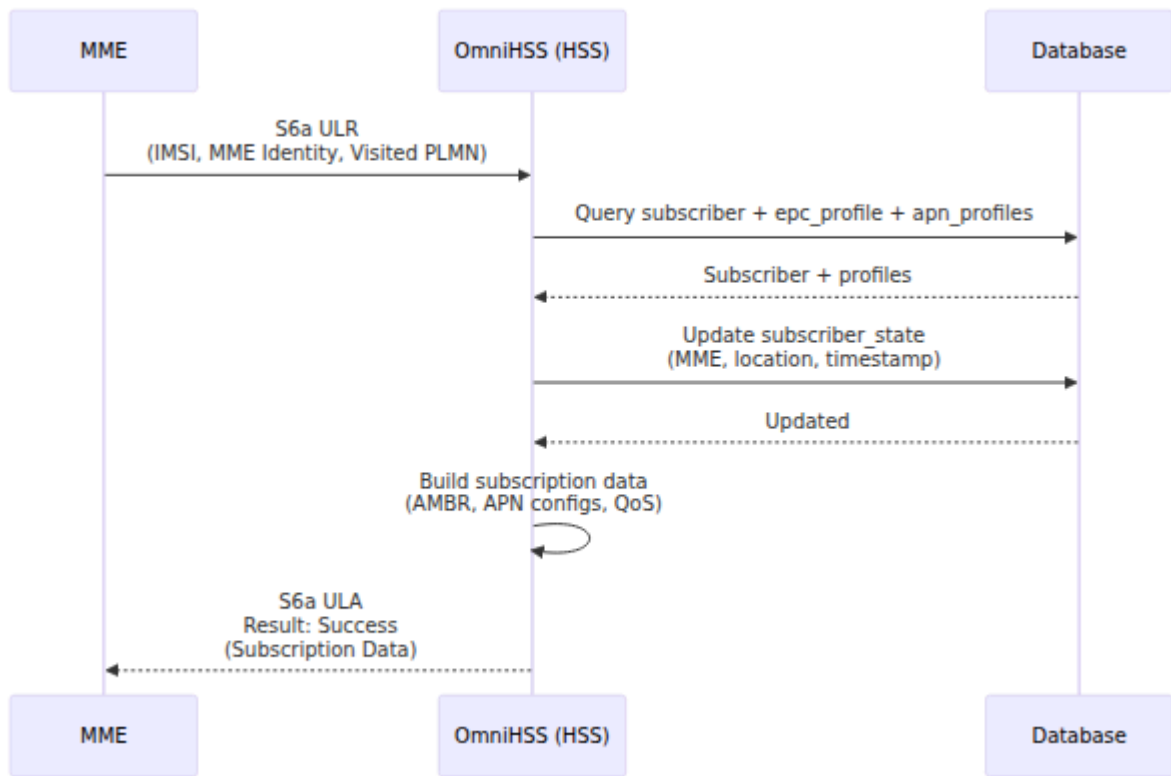


### Key AVPs:

- Request: User-Name (IMSI), RAT-Type, ULR-Flags, Visited-PLMN-Id, UE-SRVCC-Capability
- Response: Subscription-Data (AMBR, APN-Configuration, Network-Access-Mode)

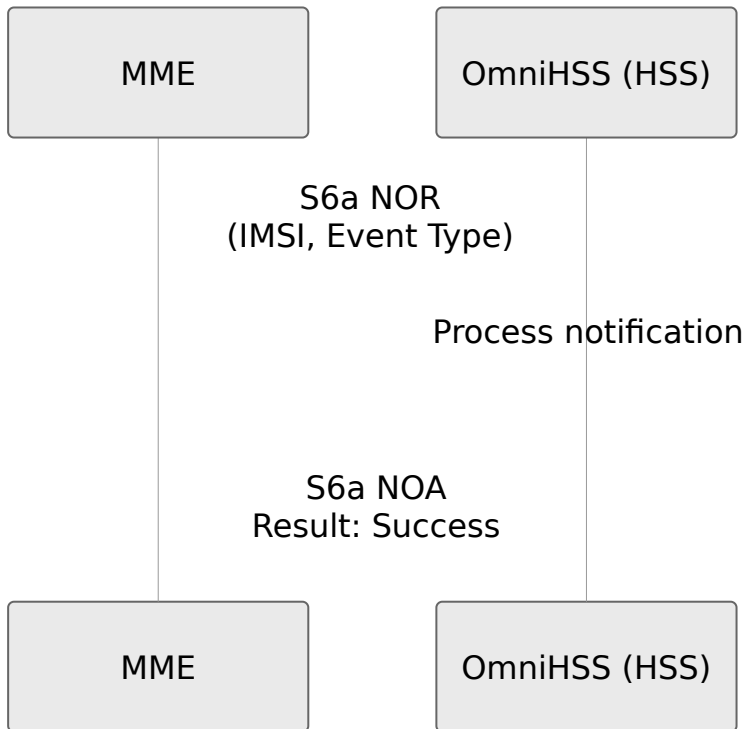
## Purge UE Request (PUR/PUA)

MME notifies HSS when subscriber context is deleted.



## Notify Request (NOR/NOA)

MME informs HSS of various events.

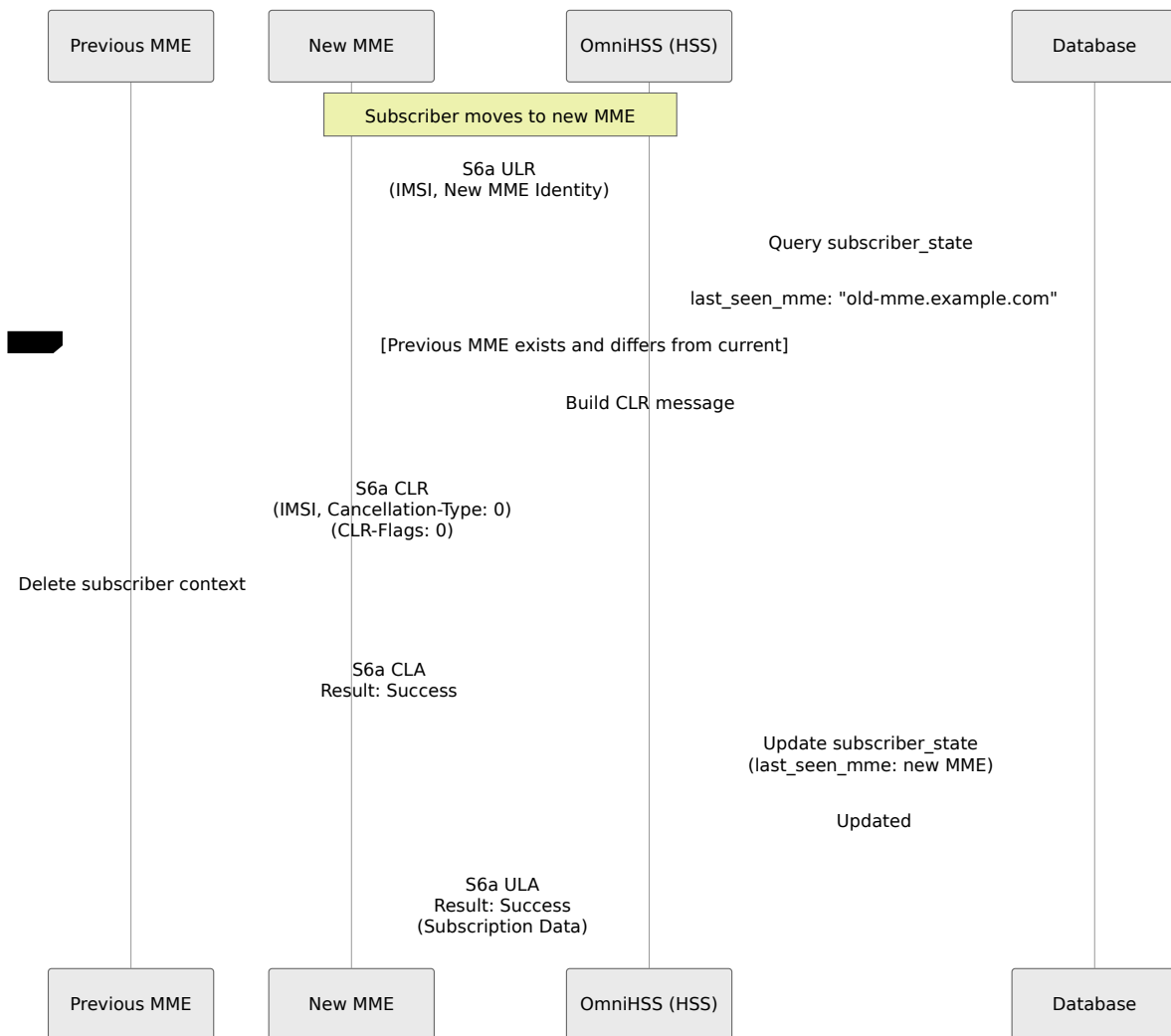


# Cancel Location Request (CLR/CLA)

HSS initiates location cancellation to inform MME that subscriber should be detached. OmniHSS supports both automatic and programmatic CLR sending.

## Automatic CLR (MME Handover)

When a subscriber performs an Update Location Request from a new MME, OmniHSS automatically sends a CLR to the previous MME to clean up stale registrations.



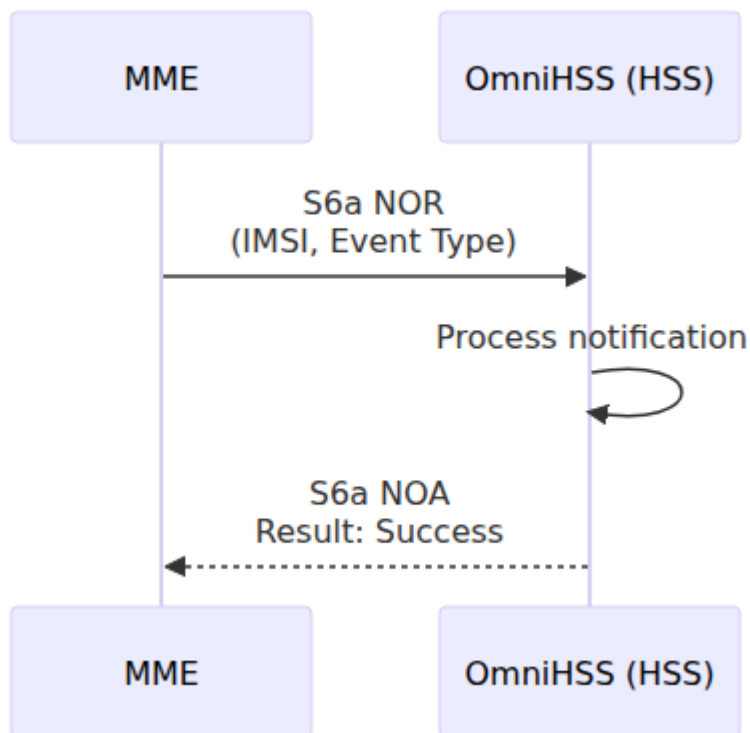
## Key AVPs (Automatic CLR):

- User-Name: IMSI of subscriber
- Destination-Host: Previous MME hostname
- Destination-Realm: Previous MME realm
- Cancellation-Type: 0 (MME Update Procedure)

- CLR-Flags: 0
- Subscription-Data: Full subscription profile

### Programmatic CLR (API-Triggered)

Administrators can trigger CLR via the programmatic API to forcibly detach subscribers (e.g., for subscription withdrawal, fraud prevention, or administrative actions).



### Key AVPs (Programmatic CLR):

- User-Name: IMSI of subscriber
- Destination-Host: Last seen MME hostname
- Destination-Realm: Last seen MME realm
- Cancellation-Type: `:subscription_withdrawal` (encoded as integer per 3GPP TS 29.272)
- CLR-Flags:
  - s6a\_indicator: 1 (indicates S6a interface)
  - reattach\_required: 1 (UE must re-authenticate to reattach)

### Cancellation Types

OmniHSS supports multiple cancellation types per 3GPP TS 29.272:

Type	Value	Description	Use Case
MME Update Procedure	0	Normal MME change	Automatic during ULR from new MME
SGSN Update Procedure	1	SGSN handover	3G/2G handover scenarios
Subscription Withdrawal	2	Admin termination	Manual detach via API
Update Procedure IWF	3	Interworking function update	Legacy network interop
Initial Attach Procedure	4	Fresh registration	Force re-authentication

## CLR-Flags

The CLR-Flags AVP is a bitmask with the following fields:

Flag	Bit	Description
S6a/S6d Indicator	0	1 = S6a interface used
Reattach Required	1	1 = UE must perform new attach

## Example CLR-Flags Configuration:

```
clr_flags: %{
  s6a_indicator: 1,          # Using S6a interface
  reattach_required: 1      # Force re-authentication
}
```

## Multi-IMSI Scenarios

OmniHSS tracks MME registration **per subscriber (IMSI)**, not per MSISDN. This is critical for understanding CLR behavior in multi-IMSI scenarios:

### Scenario 1: Multiple MSISDNs, Single IMSI

Subscriber A:

- IMSI: 999000123456789
- MSISDNs: ["+1234567890", "+9876543210"]
- last\_seen\_mme: "mme01.operator.com"

When this subscriber moves to a new MME:

- **One CLR sent** to "mme01.operator.com" with IMSI 999000123456789
- Both MSISDNs are affected (same subscriber, same SIM)
- User-Name AVP contains the IMSI, not MSISDNs

### Scenario 2: Multiple Subscribers (Different IMSIs), Same MSISDN

OmniHSS enforces **unique MSISDN constraint** (one MSISDN cannot belong to multiple subscribers simultaneously). However, during porting/migration:

Subscriber A:

- IMSI: 9990001111111111
- MSISDN: "+1234567890"
- last\_seen\_mme: "mme01.operator.com"

Subscriber B (after porting):

- IMSI: 9990002222222222
- MSISDN: "+1234567890" # Same MSISDN, different SIM/IMSI
- last\_seen\_mme: "mme02.operator.com"

When Subscriber B registers:

- **No CLR sent** (different IMSI = different subscriber)
- Subscriber A remains registered at mme01
- Subscriber B registers at mme02
- Both can be active simultaneously (different physical devices)

## Scenario 3: Programmatic CLR for Multi-MSISDN Subscriber

Result:

- **One CLR sent** to the subscriber's last\_seen\_mme
- **All MSISDNs** associated with that IMSI are effectively detached
- The IMSI is the primary key for tracking MME registration

### Important Notes

1. **IMSI is the Key:** CLR operations are always **per IMSI**, never per MSISDN. The `subscriber_state` table tracks `last_seen_mme` by subscriber (IMSI).
2. **Atomic Operation:** Each subscriber can only be registered at one MME at a time. The automatic CLR ensures this by cleaning up the old registration.
3. **No CLR if No Previous MME:** If `last_seen_mme` is `nil` (subscriber never registered), no CLR is sent during ULR.
4. **Subscription Data Included:** The automatic CLR (during ULR) includes the full `Subscription-Data` AVP to help the old MME properly clean up context.
5. **Asynchronous:** The CLR is sent asynchronously (fire-and-forget). The ULA response to the new MME does not wait for CLA from the old MME.
6. **CLA Handling:** OmniHSS receives CLA responses but currently discards them (`:discard` at line 398). This prevents message loops and is standard HSS behavior.

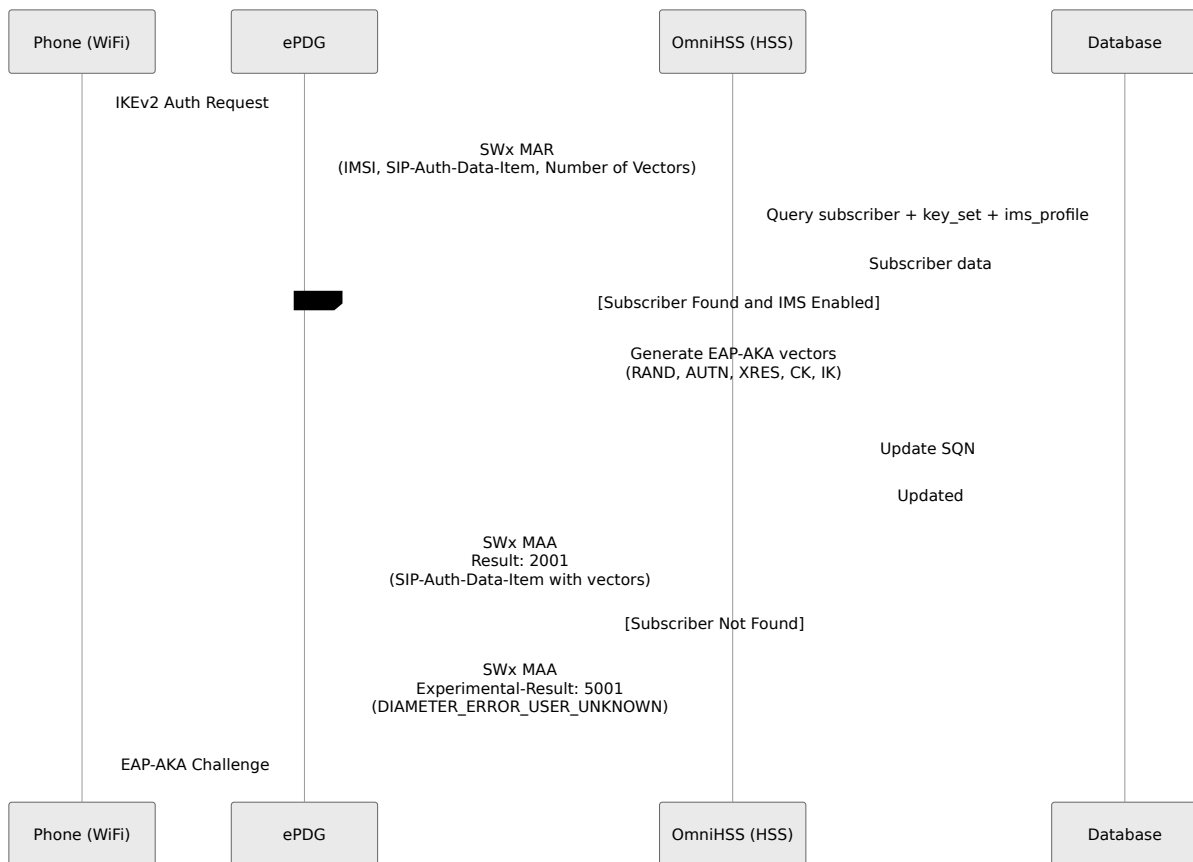
---

## SWx Interface (Non-3GPP Access / WiFi Calling)

The SWx interface enables WiFi calling by connecting the ePDG (Evolved Packet Data Gateway) to the HSS for subscriber authentication and authorization over non-3GPP access networks.

# Multimedia Auth Request (MAR/MAA)

ePDG requests EAP-AKA authentication vectors for WiFi calling registration.



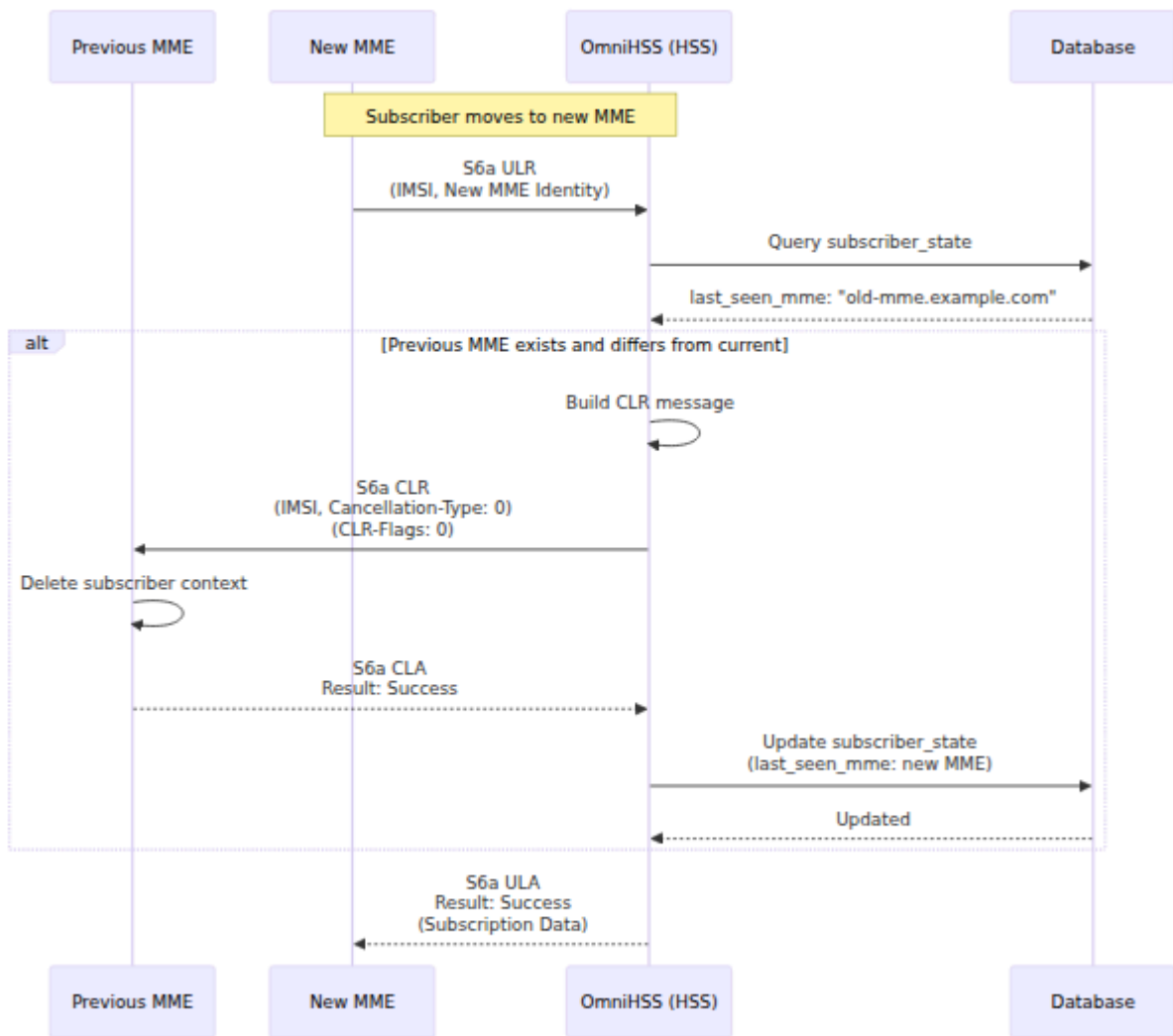
## Key AVPs:

- Request: User-Name (IMSI or IMSI@realm), SIP-Auth-Data-Item (authentication scheme), SIP-Number-Auth-Items
- Response: SIP-Auth-Data-Item (SIP-Authenticate, SIP-Authorization, Confidentiality-Key, Integrity-Key)

**SQN Resynchronization:** When the UE detects an authentication failure due to SQN mismatch, the ePDG includes resync data in the SIP-Authorization AVP. OmniHSS extracts AUTS from this field and recalculates SQN using IND-length-aware computation per 3GPP TS 33.102.

## Server Assignment Request (SAR/SAA)

ePDG registers or deregisters the subscriber for WiFi calling services.



**Server Assignment Types:**

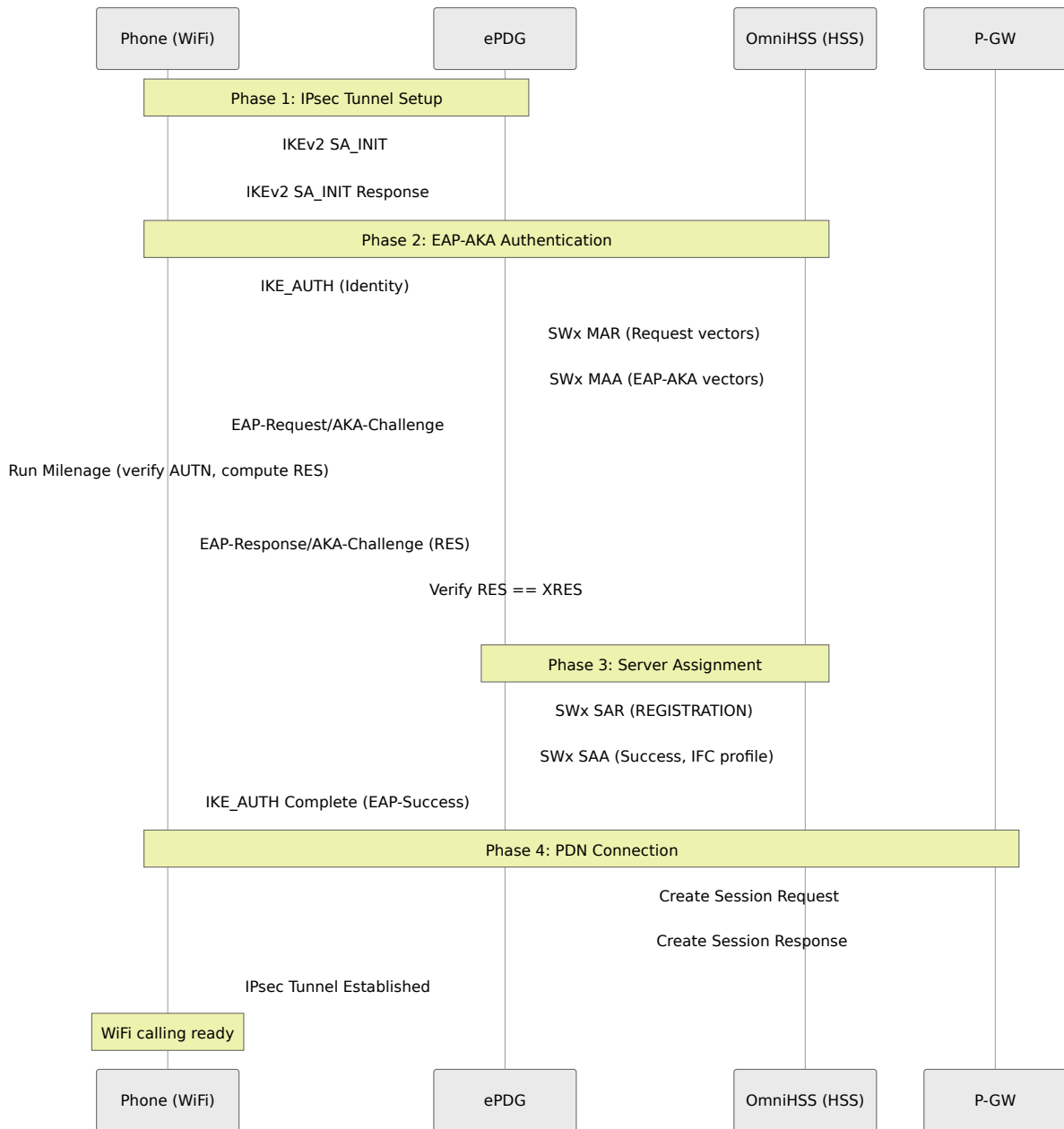
Type	Value	Description
NO_ASSIGNMENT	0	Query only, no state change
REGISTRATION	1	Initial WiFi calling registration
RE_REGISTRATION	2	Registration refresh
UNREGISTERED_USER	3	Services for unregistered user
TIMEOUT_DEREGISTRATION	4	Registration timeout
USER_DEREGISTRATION	5	User-initiated deregistration
AUTHENTICATION_FAILURE	7	Auth failed, clear state
ADMINISTRATIVE_DEREGISTRATION	8	Admin-initiated deregistration

### Key AVPs:

- Request: User-Name (IMSI), Server-Assignment-Type, Server-Name (ePDG identity), Service-Selection (APN)
- Response: User-Data (IFC profile XML), Non-3GPP-User-Data

## WiFi Calling Registration Flow

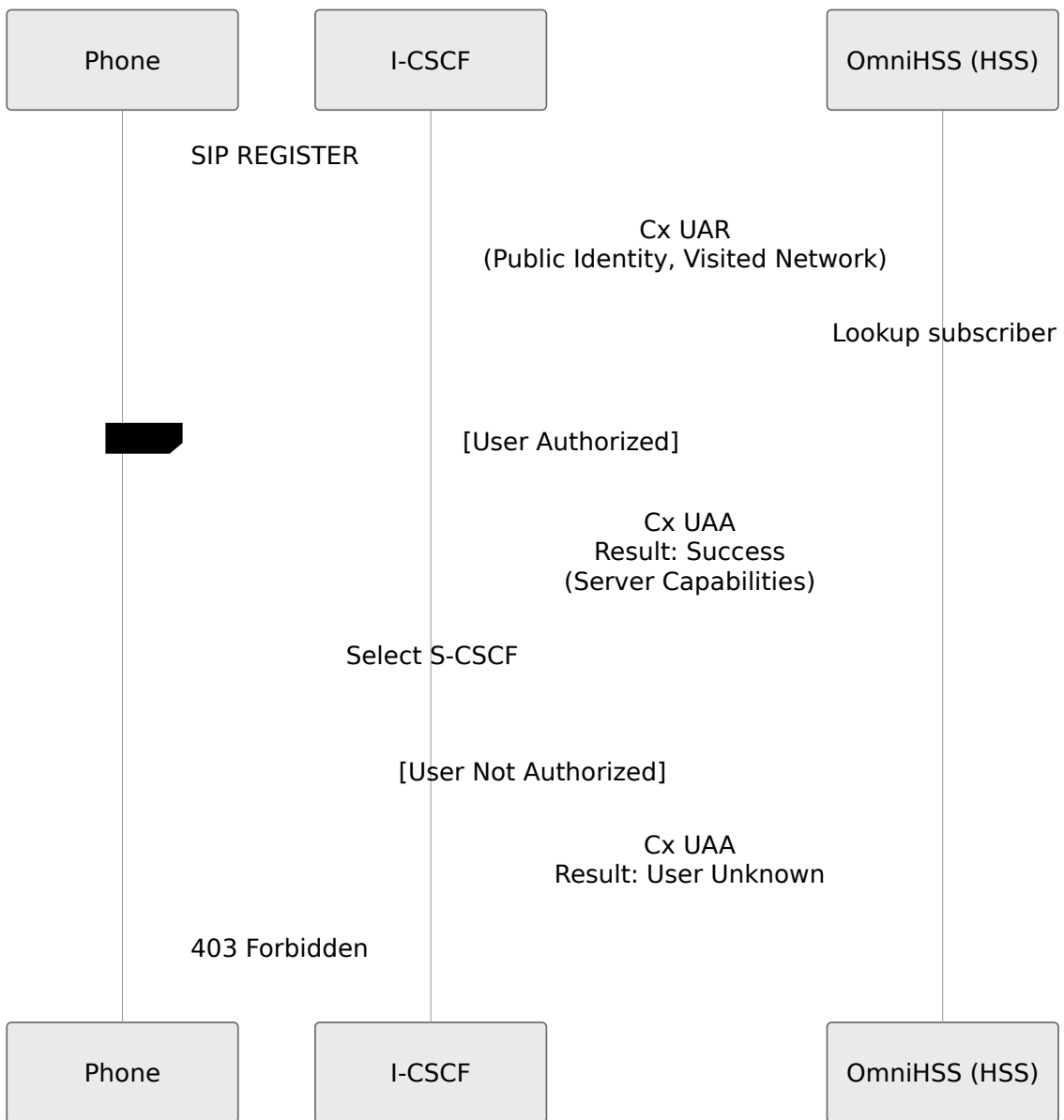
Complete end-to-end WiFi calling registration.



# Cx Interface (IMS)

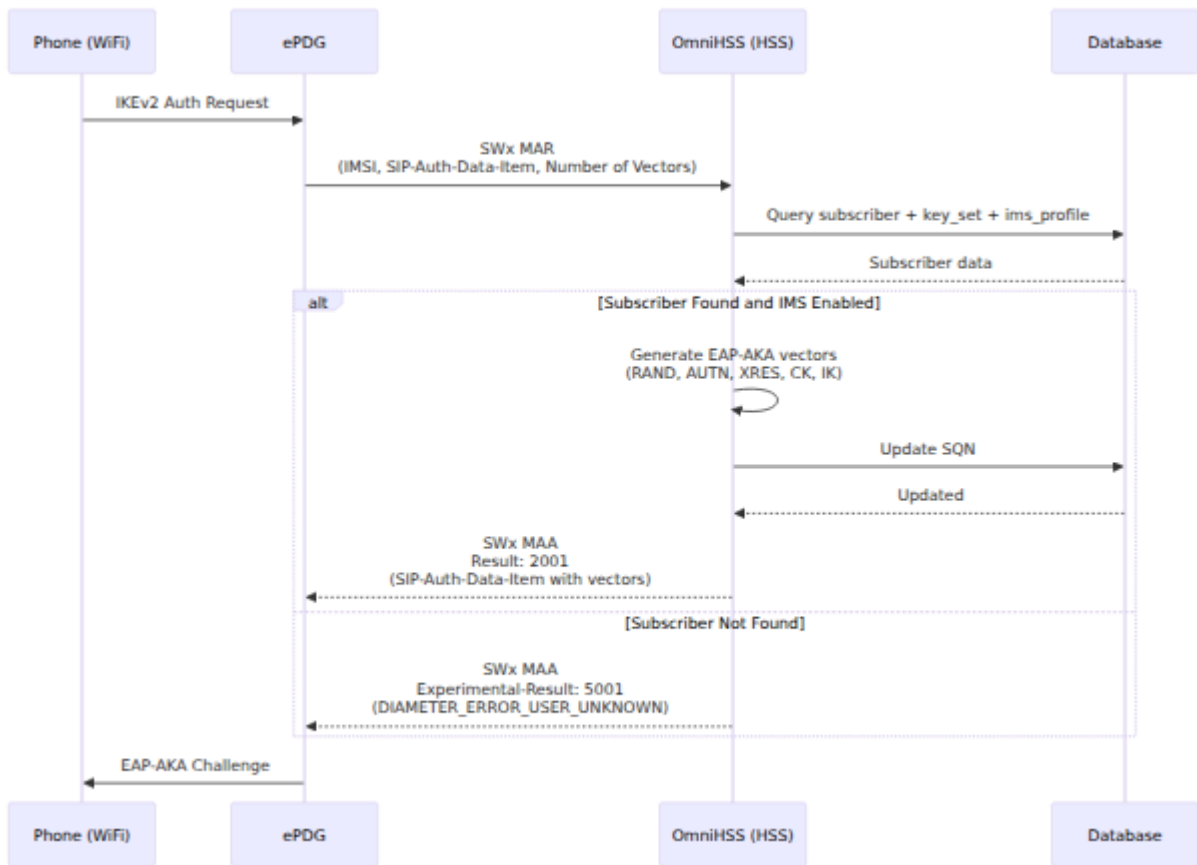
## User Authorization Request (UAR/UAA)

I-CSCF queries if user is authorized to register.



## Server Assignment Request (SAR/SAA)

S-CSCF registers/deregisters user and retrieves IMS profile.

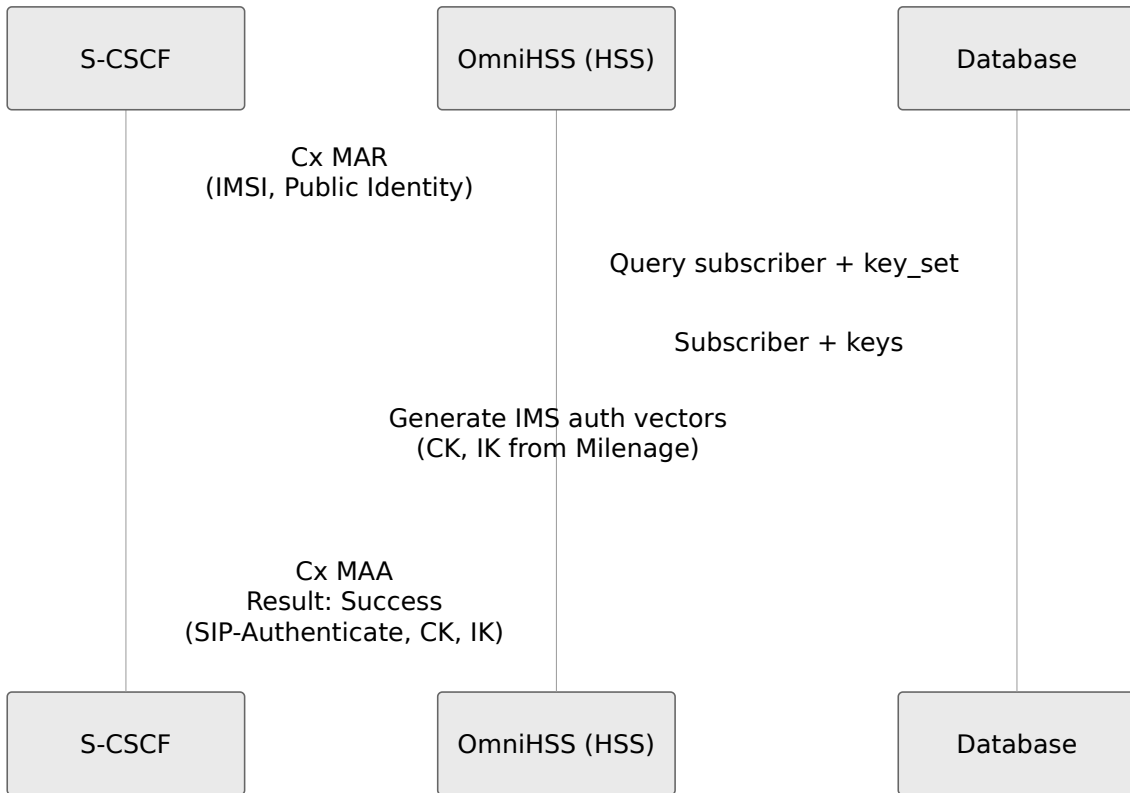


### IFC Template Rendering:

- `{{imsi}}` → Actual IMSI
- `{{msisdns}}` → List of phone numbers
- `{{mcc}}`, `{{mnc}}` → Home PLMN codes

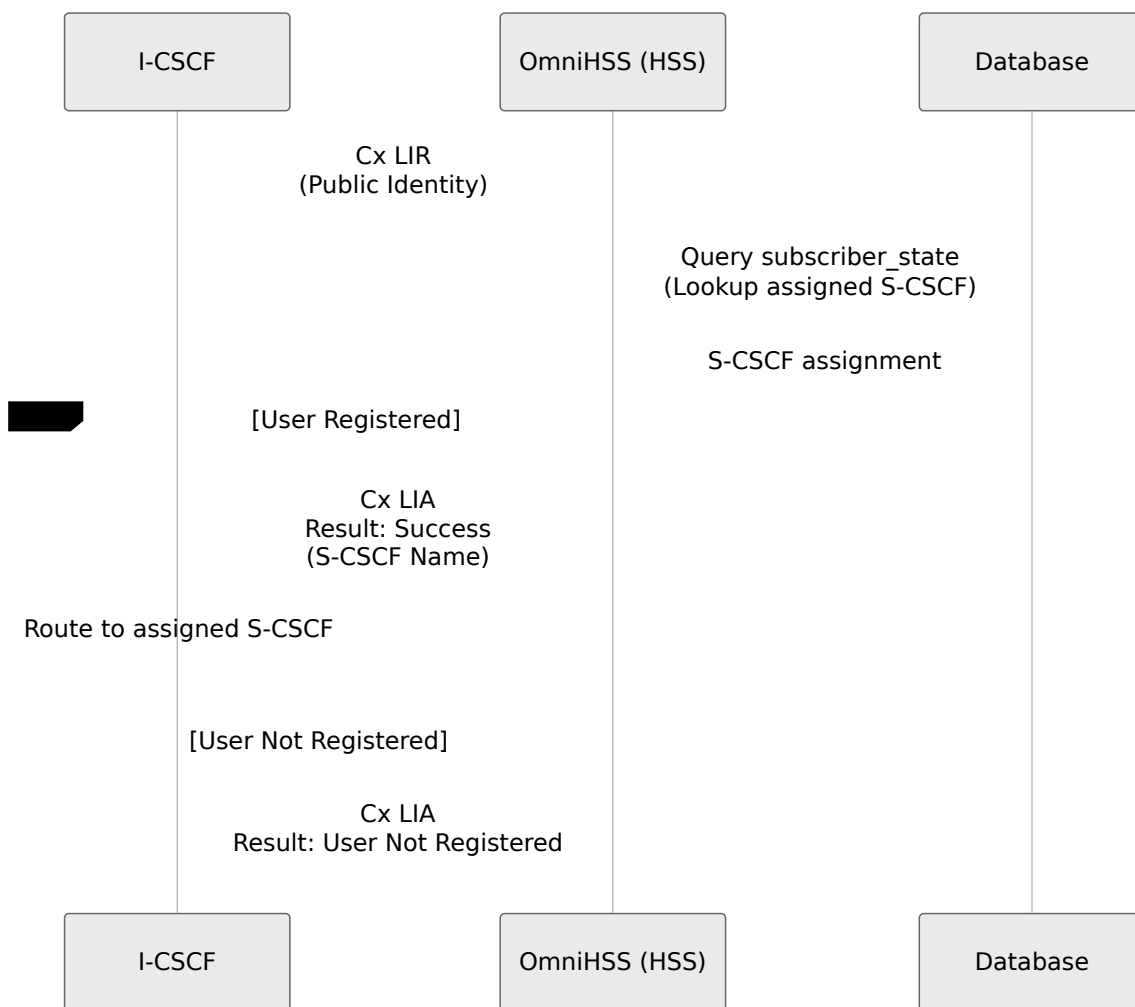
## Multimedia Auth Request (MAR/MAA)

S-CSCF requests authentication vectors for IMS registration.



## Location Info Request (LIR/LIA)

I-CSCF queries which S-CSCF is serving the user.

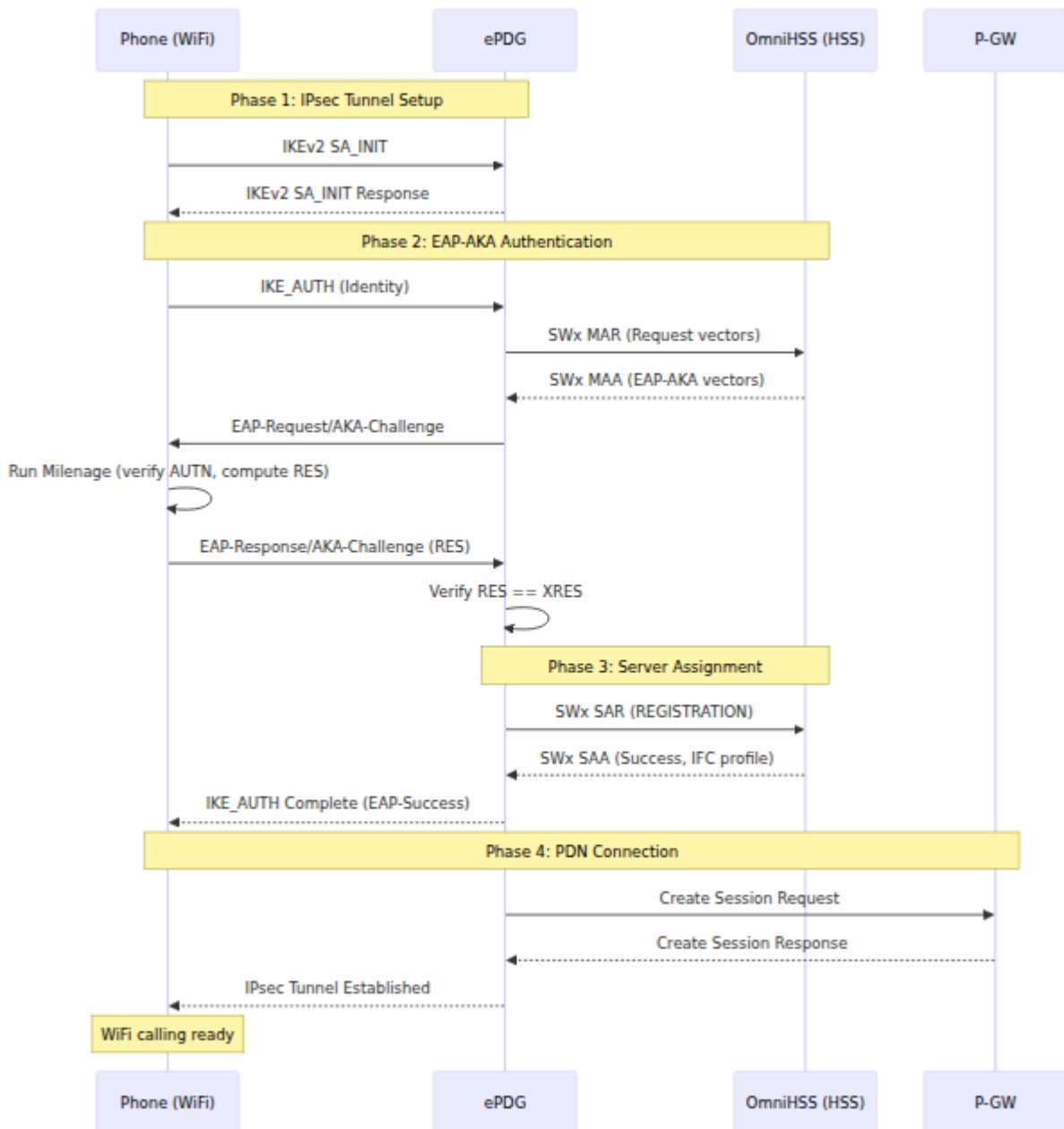


## Sh Interface (IMS Profile Data)

See [Sh RepositoryData \(Transparent Data\)](#) for the data model, SequenceNumber/concurrency rules, and the REST provisioning API for MMTEL-Services / IMS-ODB-Information documents.

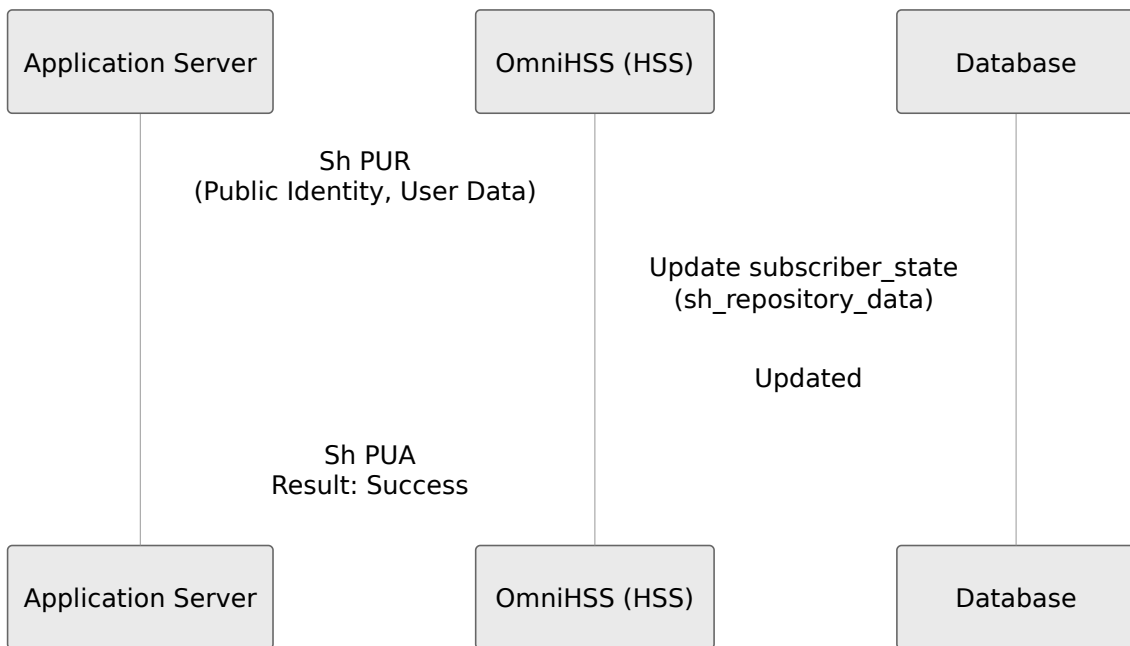
## User Data Request (UDR/UDA)

Application Server requests subscriber profile data.



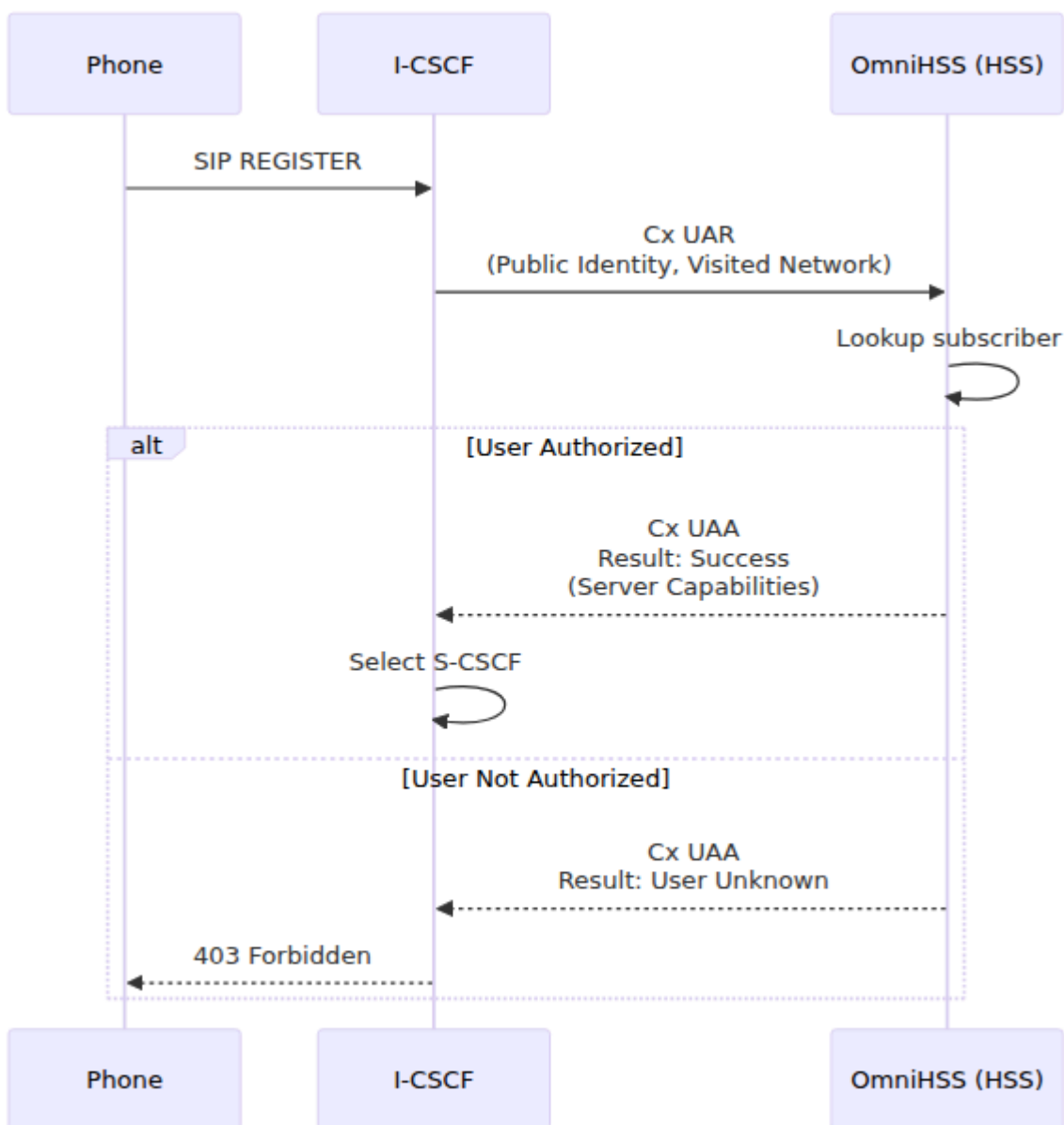
## Profile Update Request (PUR/PUA)

Application Server updates subscriber profile data.



## Subscribe Notifications Request (SNR/SNA)

Application Server subscribes to profile changes.



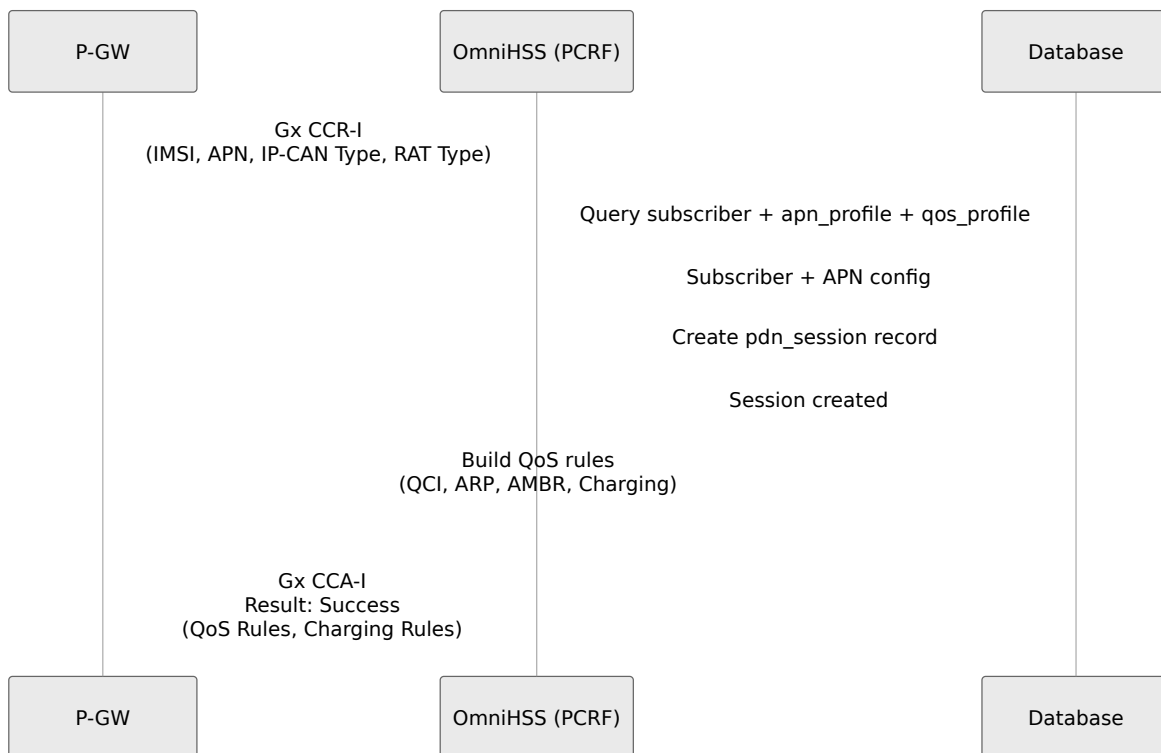
## Gx Interface (Policy Control)

OmniHSS functions as the PCRF (Policy and Charging Rules Function) via the Gx interface.

See [PCRF Documentation](#) for detailed architecture, policy configuration, and QoS management.

## Credit Control Request - Initial (CCR-I/CCA-I)

P-GW requests policy rules when PDN session is established.

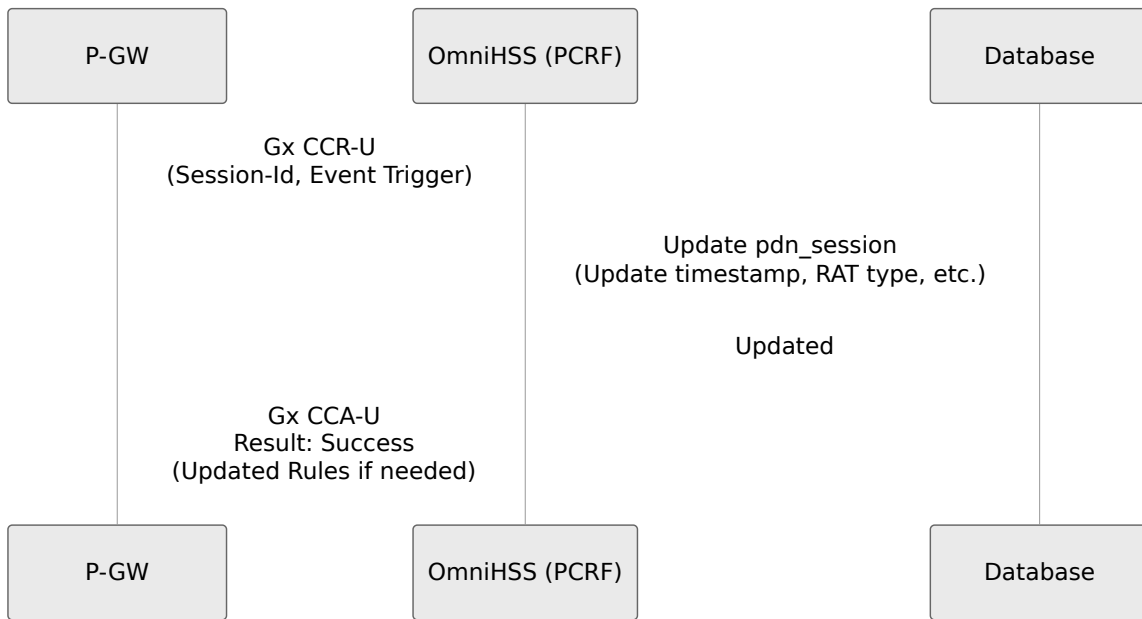


### Key AVPs:

- Request: Subscription-Id (IMSI), Called-Station-Id (APN), RAT-Type, IP-CAN-Type
- Response: QoS-Information (QCI, ARP, AMBR), Charging-Rule-Install

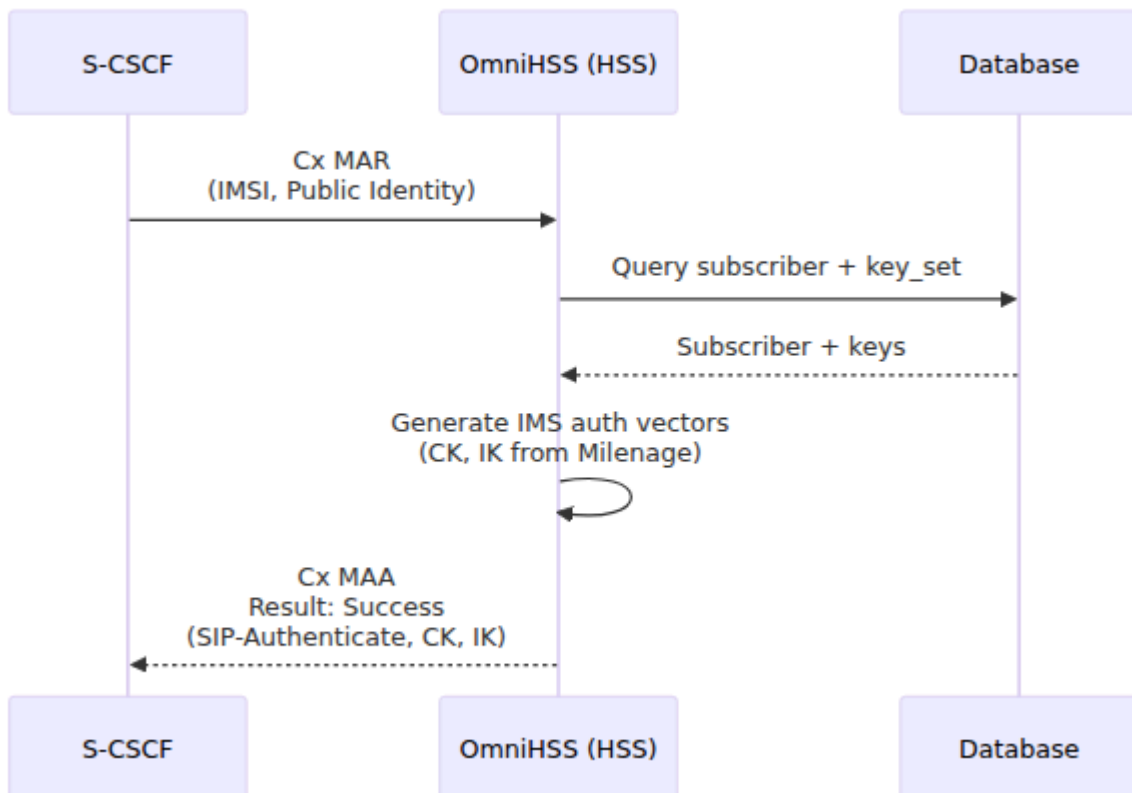
## Credit Control Request - Update (CCR-U/CCA-U)

P-GW notifies of session changes.



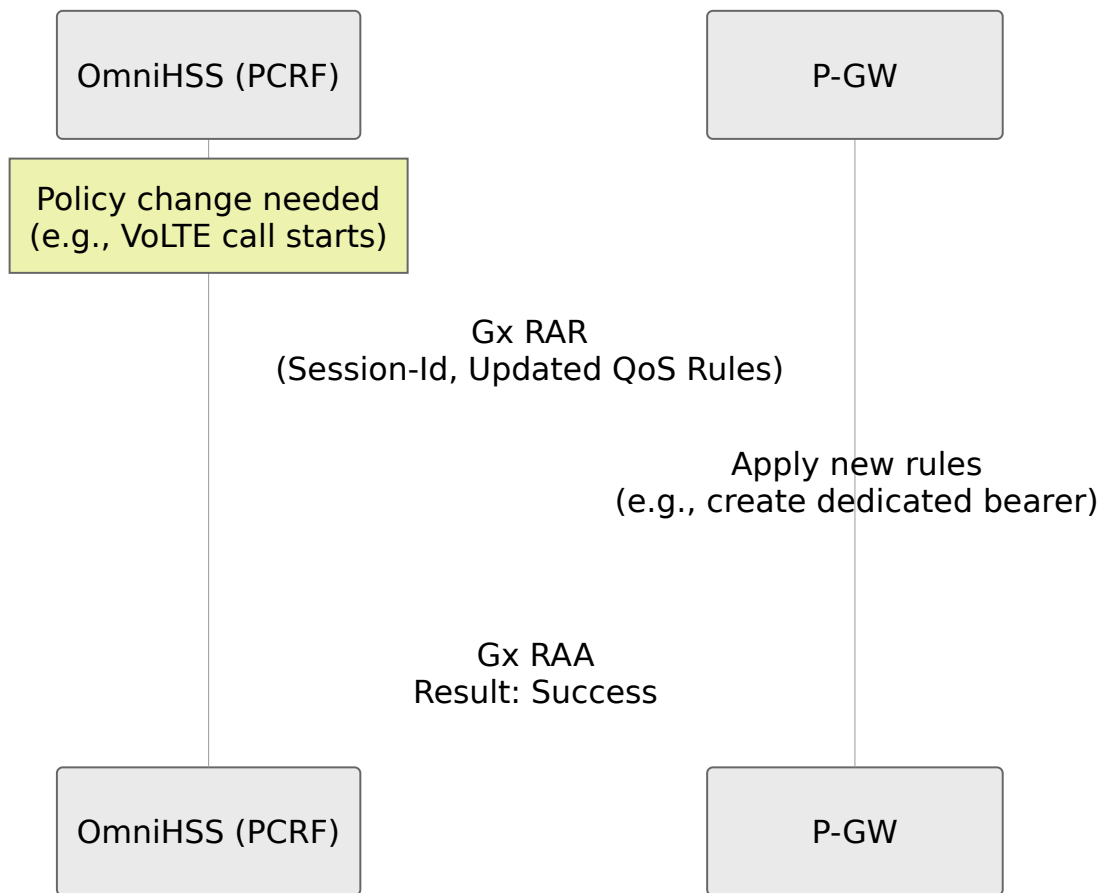
## Credit Control Request - Terminate (CCR-T/CCA-T)

P-GW notifies when PDN session ends.



## Re-Auth Request (RAR/RAA)

OmniHSS (PCRF) initiates policy update to P-GW.



---

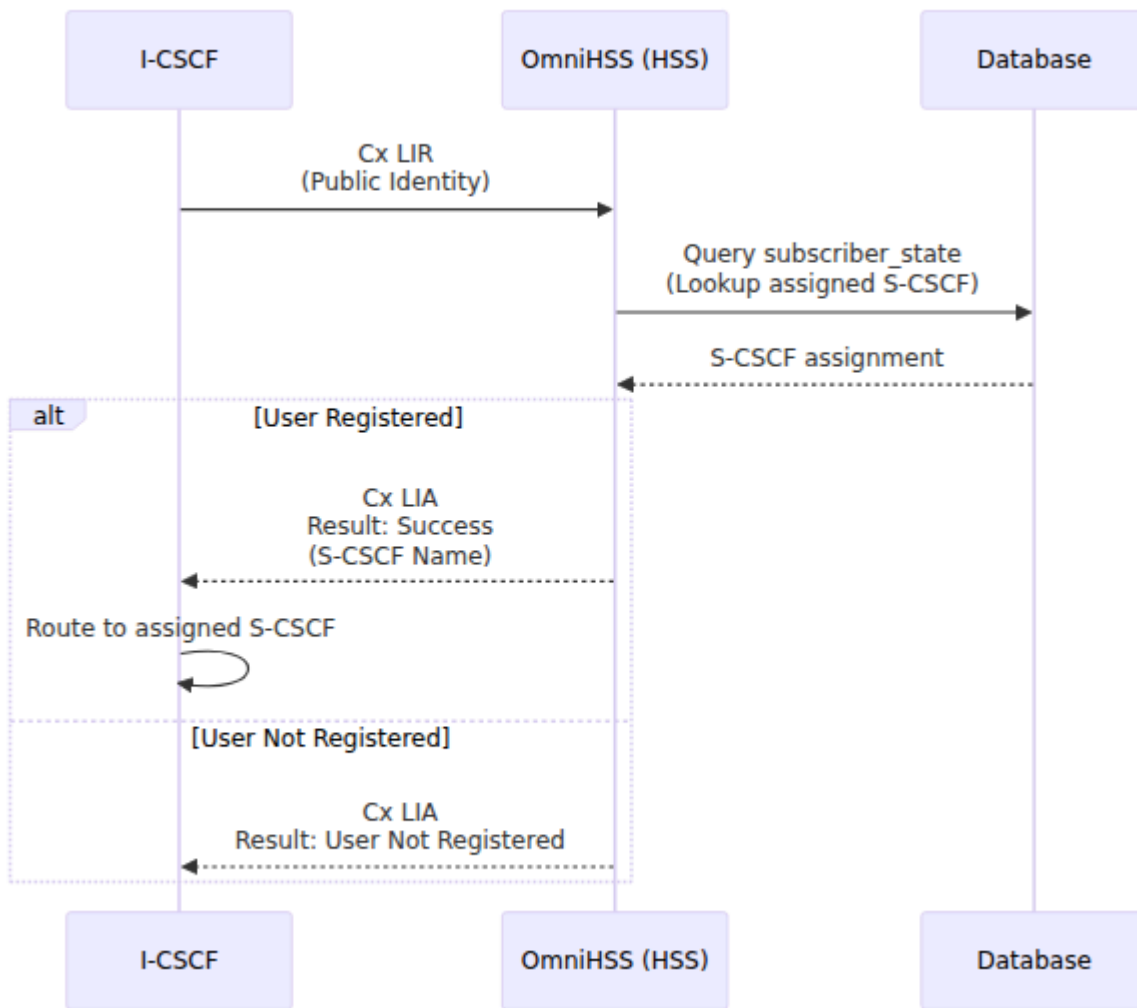
## Rx Interface (IMS Media Policy)

OmniHSS functions as the PCRF via the Rx interface for IMS media authorization.

See [PCRF Documentation](#) for detailed VoLTE call flows and media authorization.

## AA Request (AAR/AAA)

P-CSCF requests media authorization for IMS session.

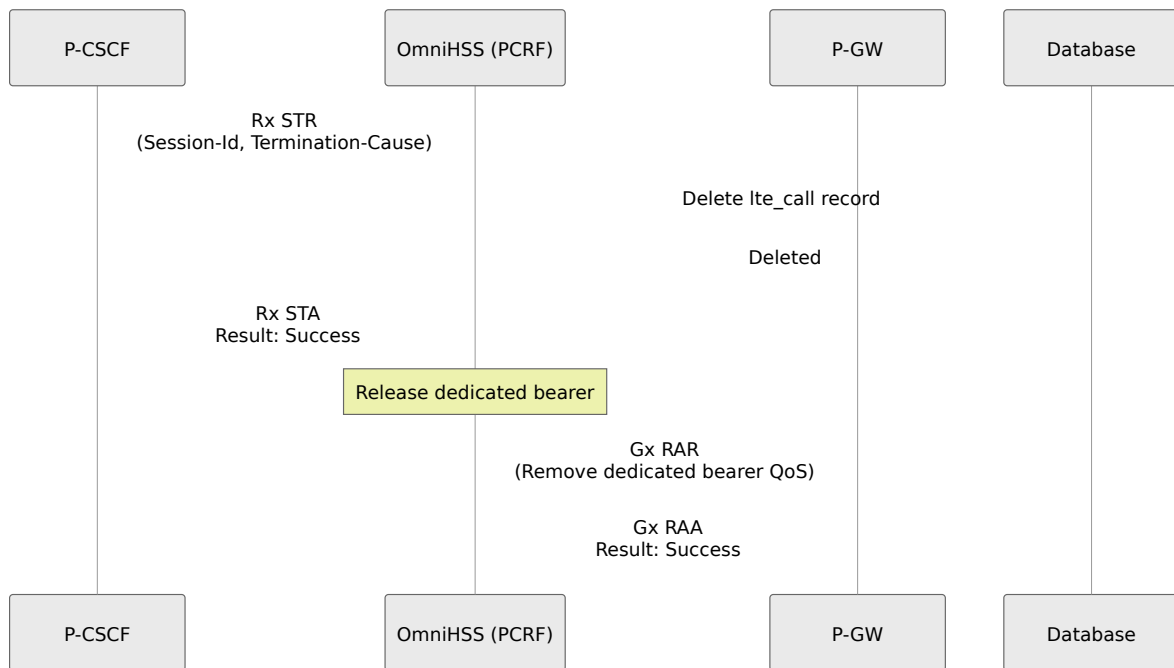


### Key Information:

- Parse SDP to determine codec and bandwidth
- Calculate required bandwidth (UL/DL)
- Create SDF filters for media flows
- Trigger dedicated bearer via Gx RAR

## Session Termination Request (STR/STA)

P-CSCF notifies when IMS session ends.



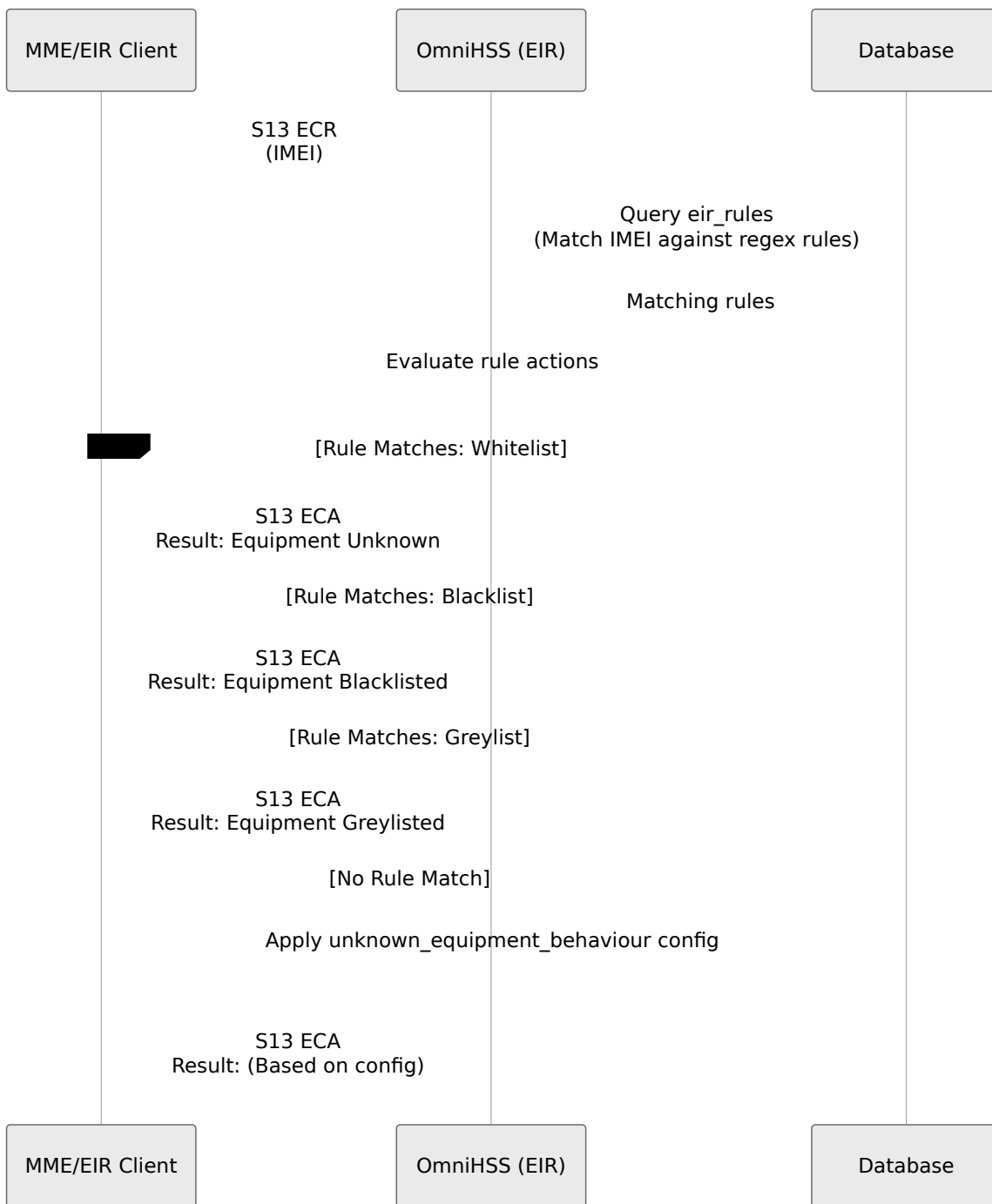
## S13 Interface (EIR)

OmniHSS functions as the EIR (Equipment Identity Register) via the S13 interface.

See [EIR Documentation](#) for detailed equipment identity checking, IMEI validation, and blacklist management.

## ME Identity Check Request (ECR/ECA)

External EIR client (or MME) requests equipment validation.



### Equipment Status Values:

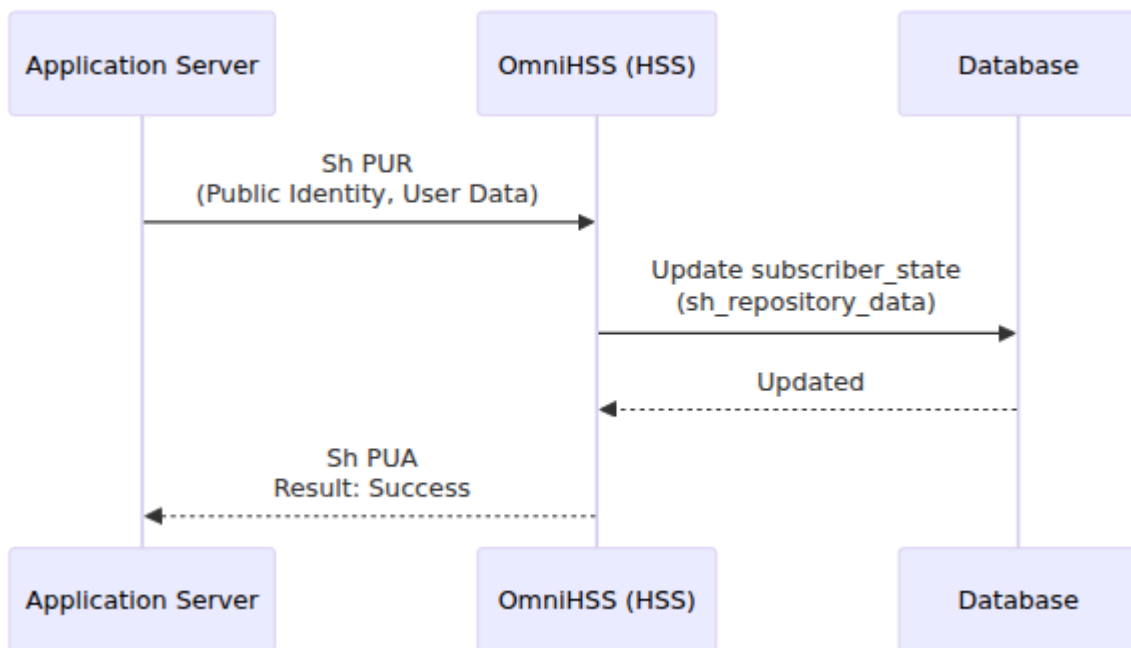
- **Equipment Unknown (0)** - Device allowed (whitelist)
- **Equipment Blacklisted (1)** - Device blocked
- **Equipment Greylisted (2)** - Device allowed but tracked

# SLh Interface (LCS / Location Services)

OmniHSS answers SLh `LCS-Routing-Info-Request` (RIR) so a GMLC can discover which MME currently serves a subscriber before requesting their location over SLg. The serving MME is the one recorded at S6a registration (Update-Location). SLh uses Diameter application id `16777291` (3GPP TS 29.173).

## LCS Routing Info Request (RIR/RIA)

GMLC requests the serving node for a subscriber.



### Notes:

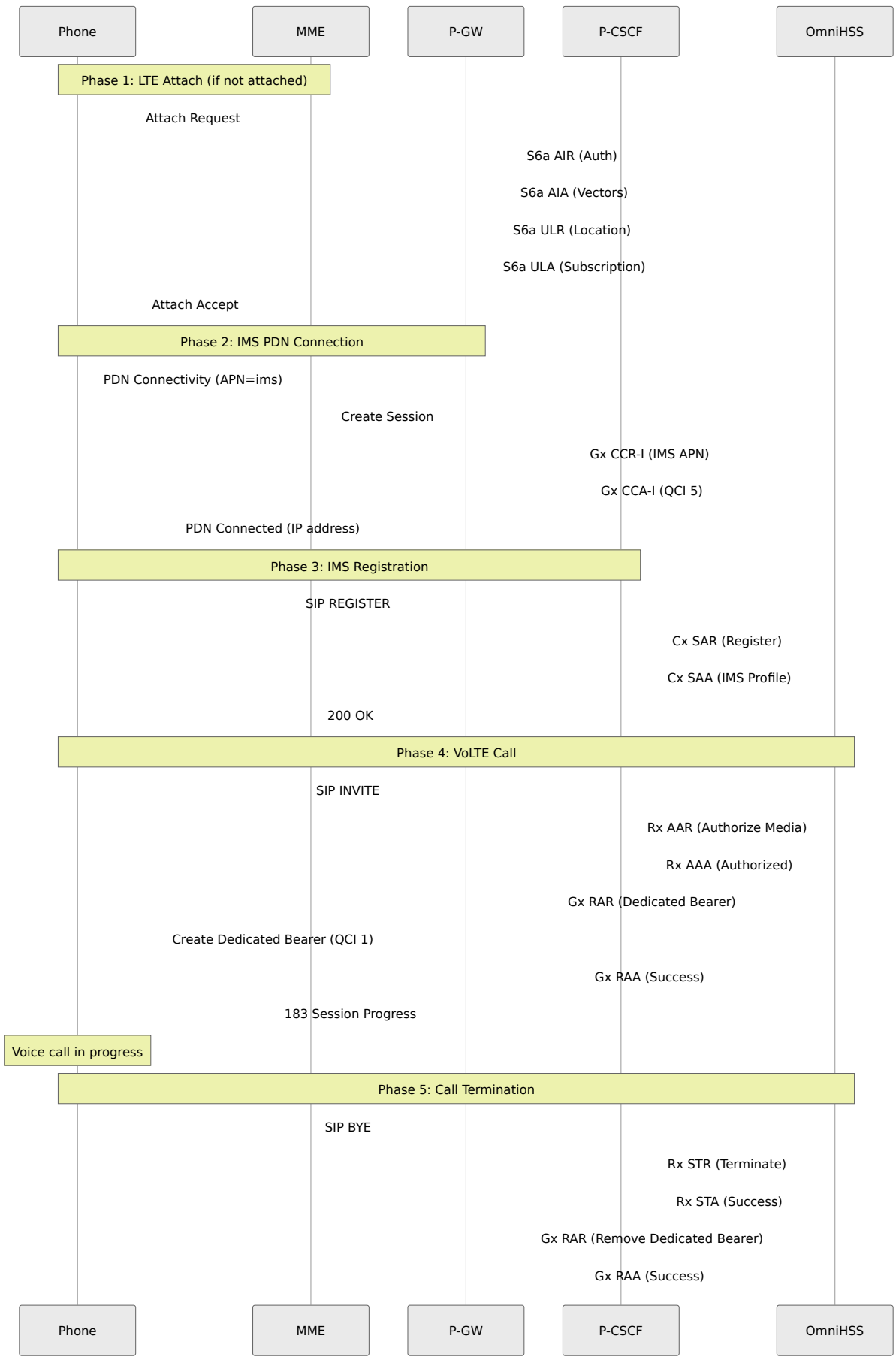
- The IMSI is taken from `User-Name`; an MSISDN lookup is used as a fallback.
- The returned `Serving-Node` carries the `MME-Name/MME-Realm` last recorded by the S6a AIR/ULR handlers (`subscriber_state.last_seen_mme / last_seen_realm`) — no extra provisioning is required.
- A GMLC typically follows a successful RIA with an SLg Provide-Location-Request to the returned MME.

### Result Codes:

- **2001** - Success; `Serving-Node` returned.
  - **5001** - `DIAMETER_ERROR_USER_UNKNOWN`; IMSI/MSISDN not provisioned.
  - **4201** - `DIAMETER_ERROR_ABSENT_USER`; subscriber known but no serving MME on record (never attached, or location cancelled).
- 

## Complete Call Flow: VoLTE Call

End-to-end VoLTE call setup showing multiple interfaces.

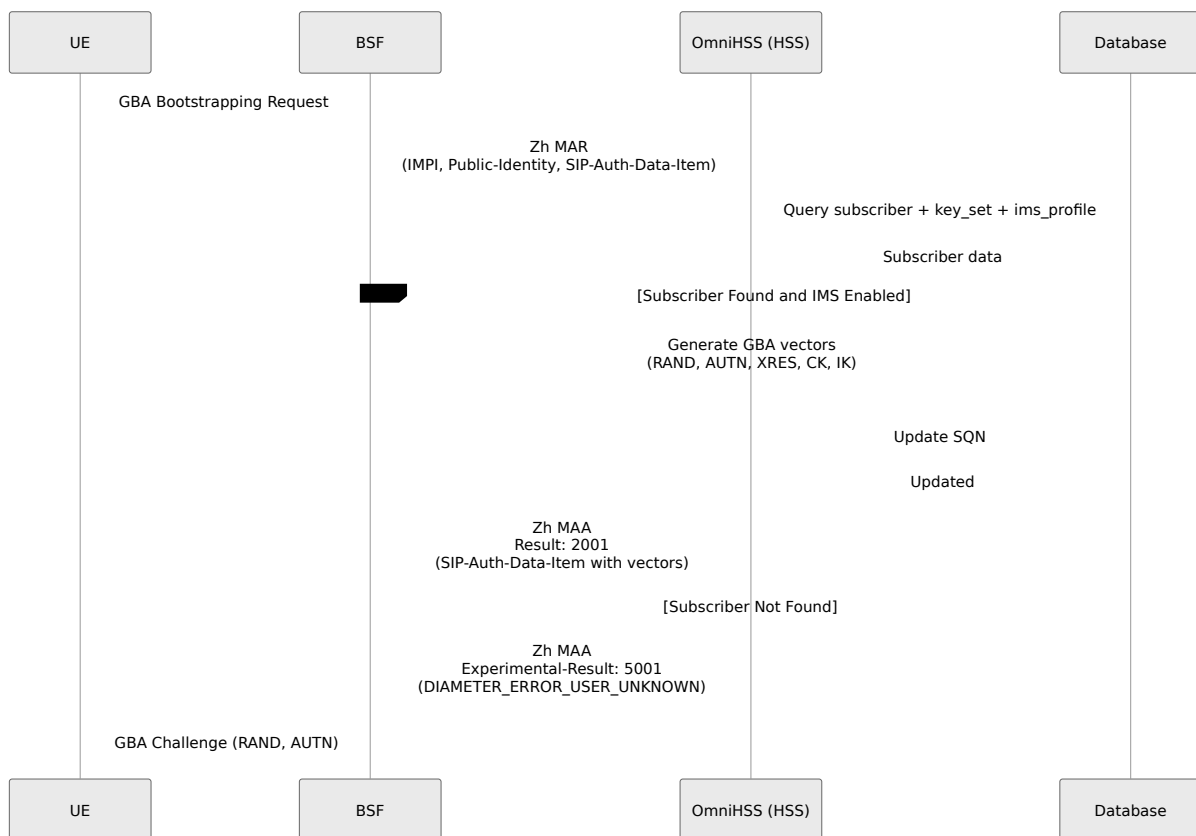


# Zh Interface (Generic Bootstrapping Architecture)

The Zh interface enables Generic Bootstrapping Architecture (GBA) by connecting the Bootstrapping Server Function (BSF) to the HSS for subscriber authentication and key derivation.

## Multimedia Auth Request (MAR/MAA)

BSF requests GBA authentication vectors for bootstrapping.



### Key AVPs:

- Request: User-Name (IMPI), Public-Identity, SIP-Auth-Data-Item (authentication scheme), Destination-Realm
- Response: SIP-Auth-Data-Item (SIP-Authenticate, SIP-Authorization, Confidentiality-Key, Integrity-Key), GBA-UserSecSettings

### GBA Use Cases:

<b>Application</b>	<b>Description</b>
MBMS	Mobile Broadcast Multicast Service key management
HTTPS	GBA-based certificate provisioning
NAF	Generic Network Application Function authentication
Presence	Secure presence service authentication

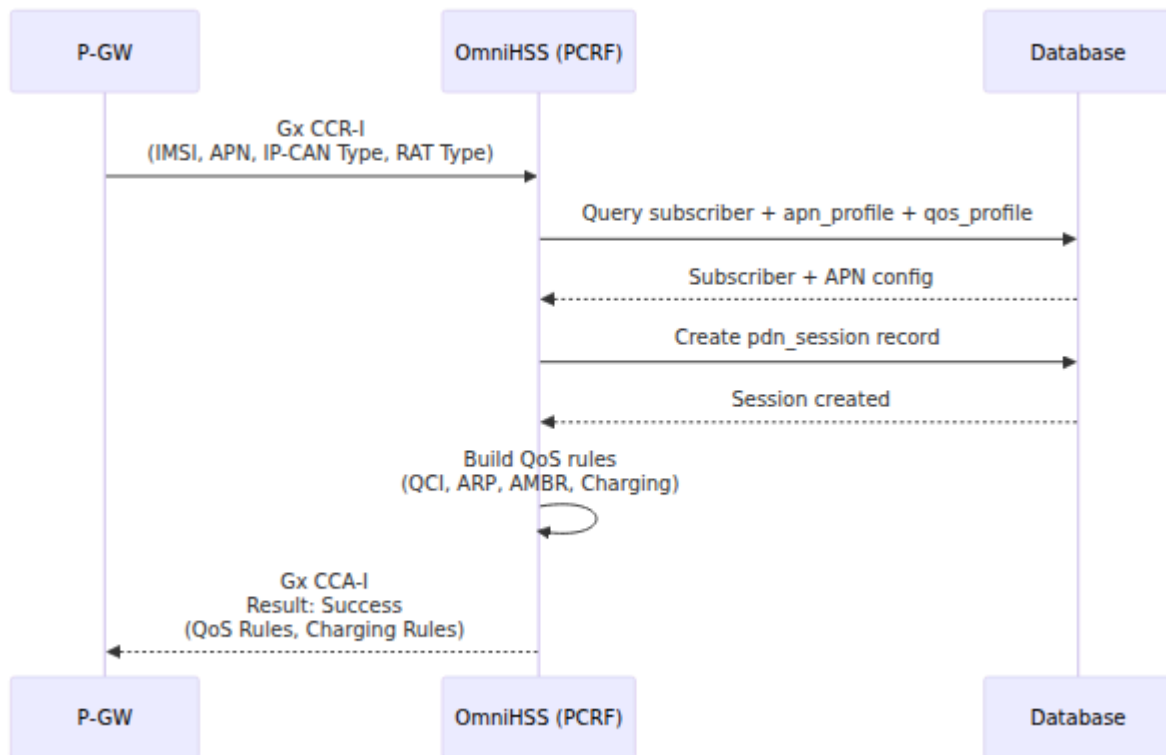
---

## **SWm Interface (Untrusted Non-3GPP Access)**

The SWm interface enables EAP-AKA authentication for untrusted non-3GPP access networks (e.g., WiFi via ePDG) by connecting the 3GPP AAA Server to the HSS.

### **Diameter-EAP-Request (DER/DEA)**

EAP authentication exchange for WiFi access.

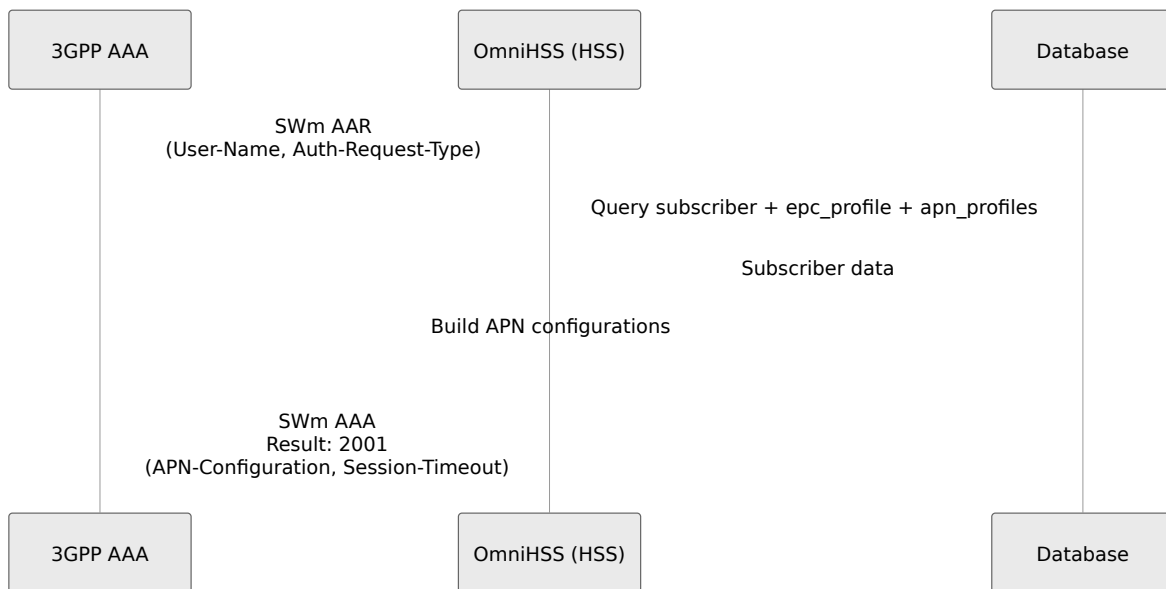


### Key AVPs:

- Request: Session-Id, Auth-Application-Id, EAP-Payload, Auth-Request-Type, User-Name (optional)
- Response: EAP-Payload, EAP-Master-Session-Key (MSK), APN-Configuration, Session-Timeout

## AA-Request (AAR/AAA)

Authorization after successful EAP authentication.

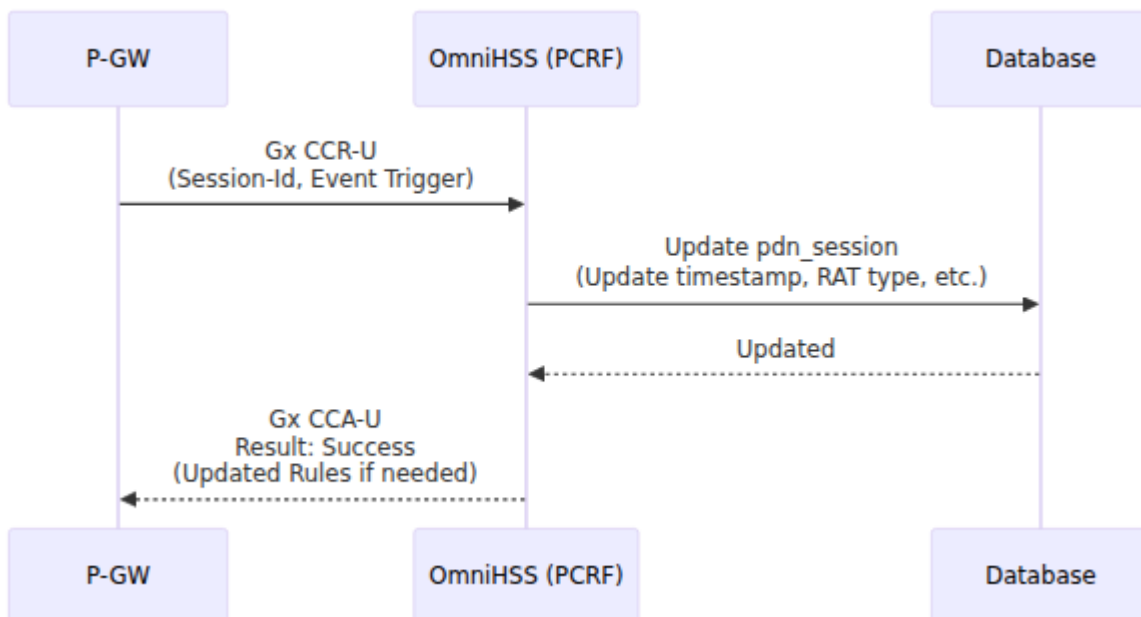


### Key AVPs:

- Request: Session-Id, Auth-Application-Id, User-Name, Auth-Request-Type, AAR-Flags
- Response: APN-Configuration, 3GPP-Charging-Characteristics, Session-Timeout

## Session-Termination-Request (STR/STA)

WiFi session termination notification.



### Termination Causes:

Cause	Value	Description
DIAMETER_LOGOUT	1	User logout
DIAMETER_SERVICE_NOT_PROVIDED	2	Service unavailable
DIAMETER_BAD_ANSWER	3	Protocol error
DIAMETER_ADMINISTRATIVE	4	Admin termination
DIAMETER_LINK_BROKEN	5	Connection lost
DIAMETER_AUTH_EXPIRED	6	Auth timeout
DIAMETER_USER_MOVED	7	User moved to other access
DIAMETER_SESSION_TIMEOUT	8	Session timeout

# Troubleshooting Protocol Issues

## Authentication Failures (S6a AIR)

### Check:

1. Key set configured correctly (Ki, OPC, AMF)
2. SQN synchronization (if repeated failures)
3. Roaming rules allow visited network

## Location Update Failures (S6a ULR)

### Check:

1. EPC profile exists and has APNs configured
2. Roaming allowed for data services

3. MME identity format correct

## **IMS Registration Failures (Cx SAR)**

### **Check:**

1. IMS profile assigned to subscriber
2. IFC template valid XML
3. S-CSCF selection configured
4. MSISDNs assigned if used in template

## **PDN Connection Failures (Gx CCR-I)**

### **Check:**

1. APN exists in EPC profile's APN list
2. APN QoS profile configured
3. PDN session table not full (if limits exist)

---

[← Back to Operations Guide](#)

# OmniHSS Roaming Control

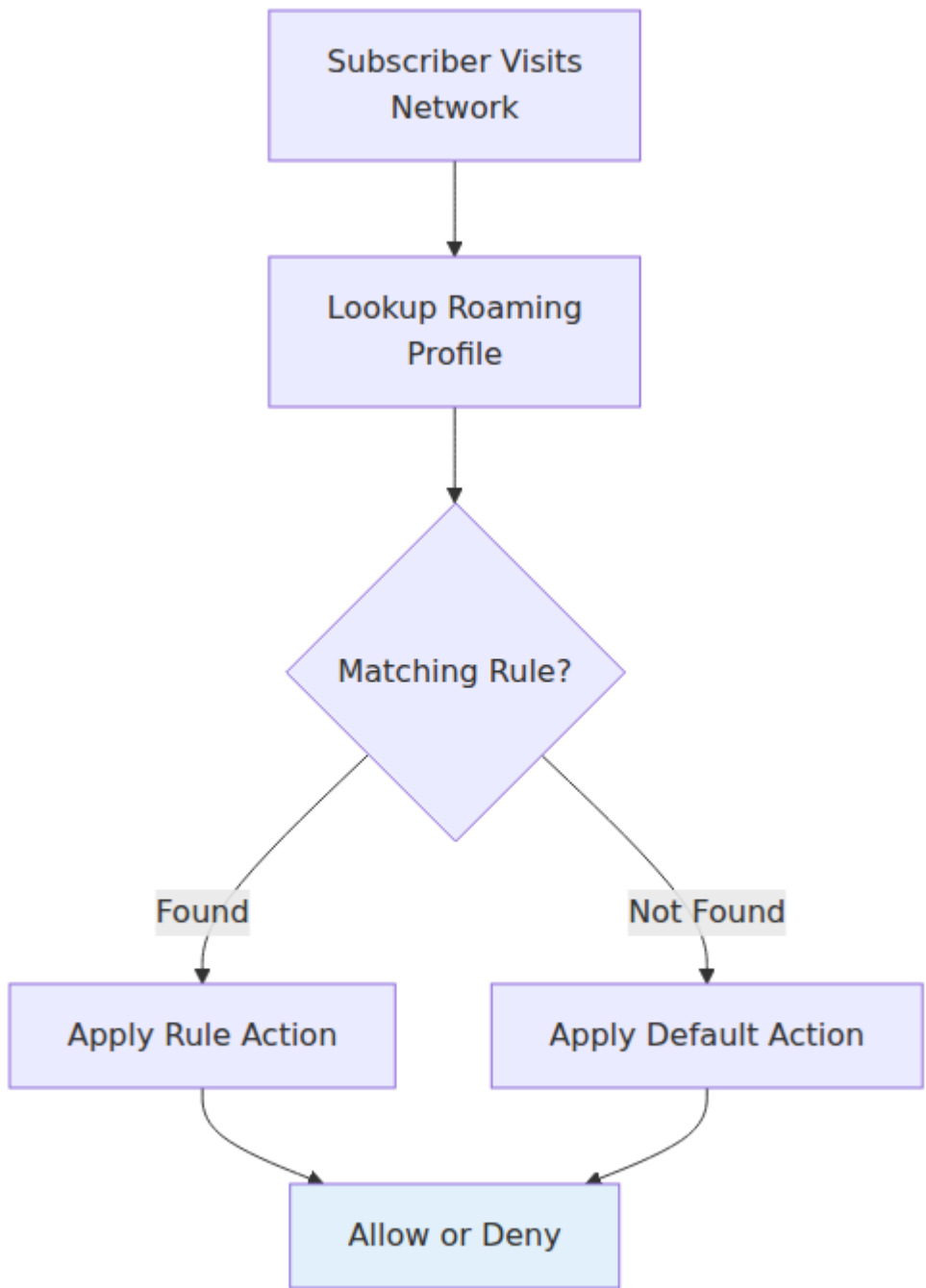
[← Back to Operations Guide](#)

---

## Overview

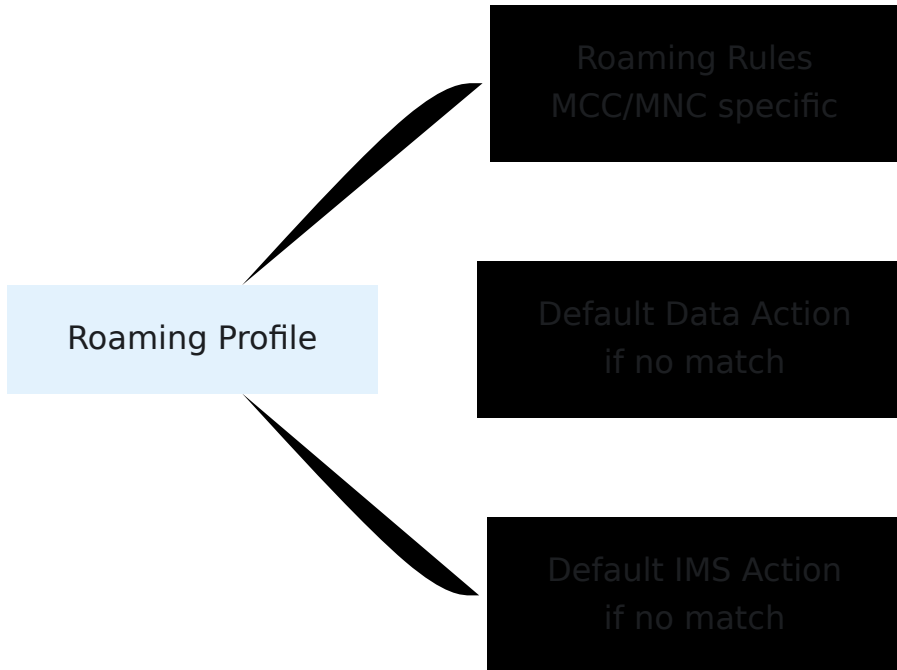
OmniHSS provides granular roaming control, allowing you to define which networks subscribers can access for both data and IMS services when roaming.

# Roaming Control Flow



# Roaming Profile Structure

## Components



## Roaming Rule

Each rule specifies action for a specific network (MCC/MNC combination).

### Fields:

- `name` - Descriptive name
- `mcc` - Mobile Country Code (3 digits)
- `mnc` - Mobile Network Code (2-3 digits)
- `data_action` - "allow" or "deny"
- `ims_action` - "allow" or "deny"

## Roaming Profile

Defines default behavior and links to rules.

### Fields:

- `name` - Profile name
  - `data_action_if_no_rules_match` - "allow" or "deny"
  - `ims_action_if_no_rules_match` - "allow" or "deny"
- 

# Configuration Examples

## Allow All Roaming

```
# Create profile that allows everything
curl -k -X POST https://hss.example.com:8443/api/roaming/profile \
  -H "Content-Type: application/json" \
  -d '{
    "roaming_profile": {
      "name": "Allow All",
      "data_action_if_no_rules_match": "allow",
      "ims_action_if_no_rules_match": "allow",
      "roaming_rules": []
    }
  }'
```

## Deny All Roaming

```
# Create profile that blocks everything
curl -k -X POST https://hss.example.com:8443/api/roaming/profile \
  -H "Content-Type: application/json" \
  -d '{
    "roaming_profile": {
      "name": "No Roaming",
      "data_action_if_no_rules_match": "deny",
      "ims_action_if_no_rules_match": "deny",
      "roaming_rules": []
    }
  }'
```

## Allow Specific Networks (Whitelist)

```
# Create AT&T rule
RULE1=$(curl -k -X POST
https://hss.example.com:8443/api/roaming/rule \
-H "Content-Type: application/json" \
-d '{
  "roaming_rule": {
    "name": "Allow AT&T",
    "mcc": "310",
    "mnc": "410",
    "data_action": "allow",
    "ims_action": "allow"
  }
}' | jq -r '.response.id')

# Create Verizon rule
RULE2=$(curl -k -X POST
https://hss.example.com:8443/api/roaming/rule \
-H "Content-Type: application/json" \
-d '{
  "roaming_rule": {
    "name": "Allow Verizon",
    "mcc": "311",
    "mnc": "480",
    "data_action": "allow",
    "ims_action": "allow"
  }
}' | jq -r '.response.id')

# Create profile with deny-by-default and link rules
curl -k -X POST https://hss.example.com:8443/api/roaming/profile \
-H "Content-Type: application/json" \
-d "{
  \"roaming_profile\": {
    \"name\": \"US Carriers Only\",
    \"data_action_if_no_rules_match\": \"deny\",
    \"ims_action_if_no_rules_match\": \"deny\",
    \"roaming_rules\": [$RULE1, $RULE2]
  }
}"
```

## Allow Data, Block Voice

```
# Create rule that allows data but blocks IMS
curl -k -X POST https://hss.example.com:8443/api/roaming/rule \
  -H "Content-Type: application/json" \
  -d '{
    "roaming_rule": {
      "name": "Data Only - T-Mobile",
      "mcc": "310",
      "mnc": "260",
      "data_action": "allow",
      "ims_action": "deny"
    }
  }'
```

## Block Specific Networks (Blacklist)

```
# Create expensive network blocking rule
RULE=$(curl -k -X POST
https://hss.example.com:8443/api/roaming/rule \
-H "Content-Type: application/json" \
-d '{
  "roaming_rule": {
    "name": "Block Expensive Network",
    "mcc": "206",
    "mnc": "01",
    "data_action": "deny",
    "ims_action": "deny"
  }
}' | jq -r '.response.id')

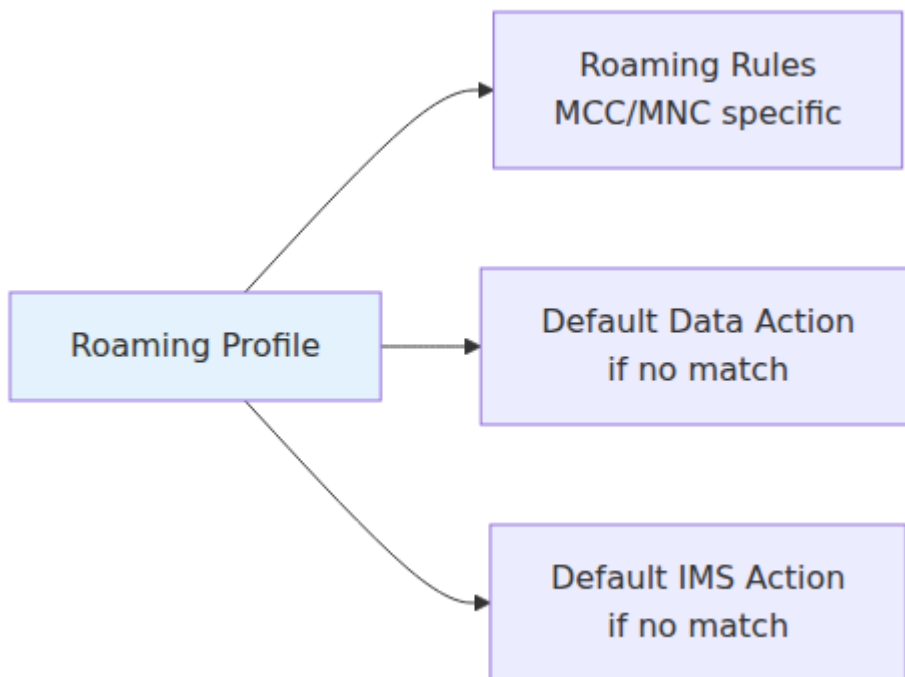
# Create profile with allow-by-default
curl -k -X POST https://hss.example.com:8443/api/roaming/profile \
-H "Content-Type: application/json" \
-d "{
  \"roaming_profile\": {
    \"name\": \"Block Expensive Networks\",
    \"data_action_if_no_rules_match\": \"allow\",
    \"ims_action_if_no_rules_match\": \"allow\",
    \"roaming_rules\": [$RULE]
  }
}"
```

---

## Common Roaming Scenarios

### Scenario 1: Domestic Roaming Only

Subscriber can roam within home country but not internationally.

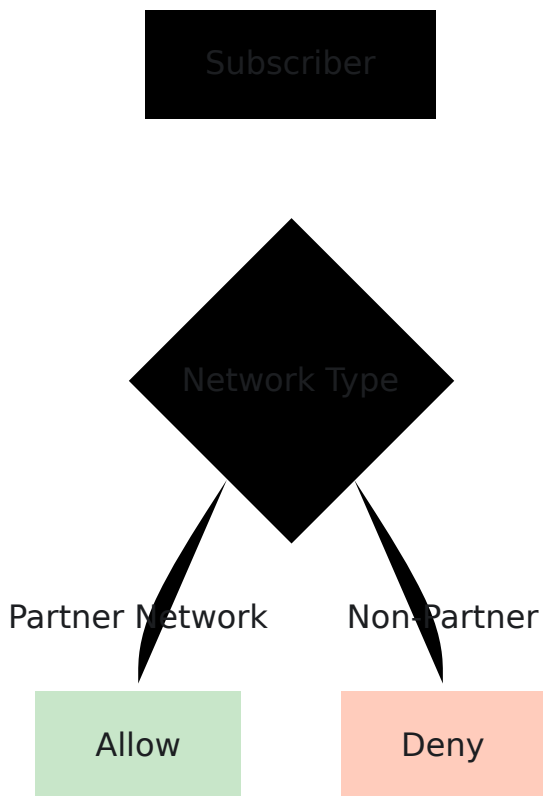


### **Configuration:**

- Default: Deny all
- Rules: Allow all USA MCC codes (310, 311, 312, 313, 314, 315, 316)

## **Scenario 2: Roaming Partners Only**

Subscriber can only roam on networks with commercial agreements.

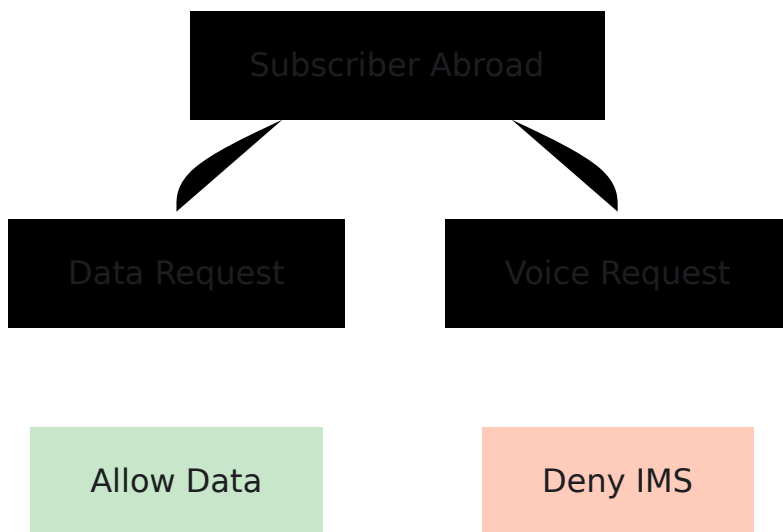


**Configuration:**

- Default: Deny all
- Rules: Allow each partner network explicitly (by MCC/MNC)

### Scenario 3: Data Roaming, No Voice Roaming

Subscriber can use data abroad but must use WiFi for voice calls.



**Configuration:**

- Rules: `data_action: "allow"`, `ims_action: "deny"`

## Scenario 4: Emergency Service Access

Always allow emergency services, even if roaming is blocked.

**Note:** Emergency service handling is typically done at the MME/network level. OmniHSS roaming rules apply to normal services.

---

## MCC/MNC Reference

### Common Country Codes (MCC)

MCC	Country	Networks
310-316	USA	AT&T, Verizon, T-Mobile, etc.
302	Canada	Rogers, Bell, Telus
234-235	United Kingdom	Vodafone, O2, EE
262	Germany	Deutsche Telekom, Vodafone
208	France	Orange, SFR, Bouygues
222	Italy	TIM, Vodafone, Wind
214	Spain	Movistar, Vodafone

## Common US Carriers (MCC 310-316)

MCC	MNC	Carrier
310	410	AT&T
311	480	Verizon
310	260	T-Mobile
310	120	Sprint
313	380	(Example test network)

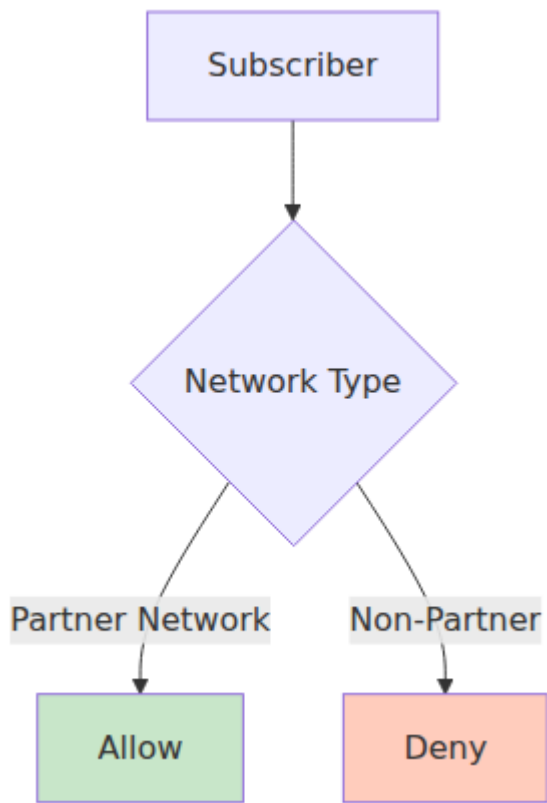
**Full Lists:** See [ITU-T E.212](#) or [MCC/MNC databases](#)

---

## Roaming Enforcement Points

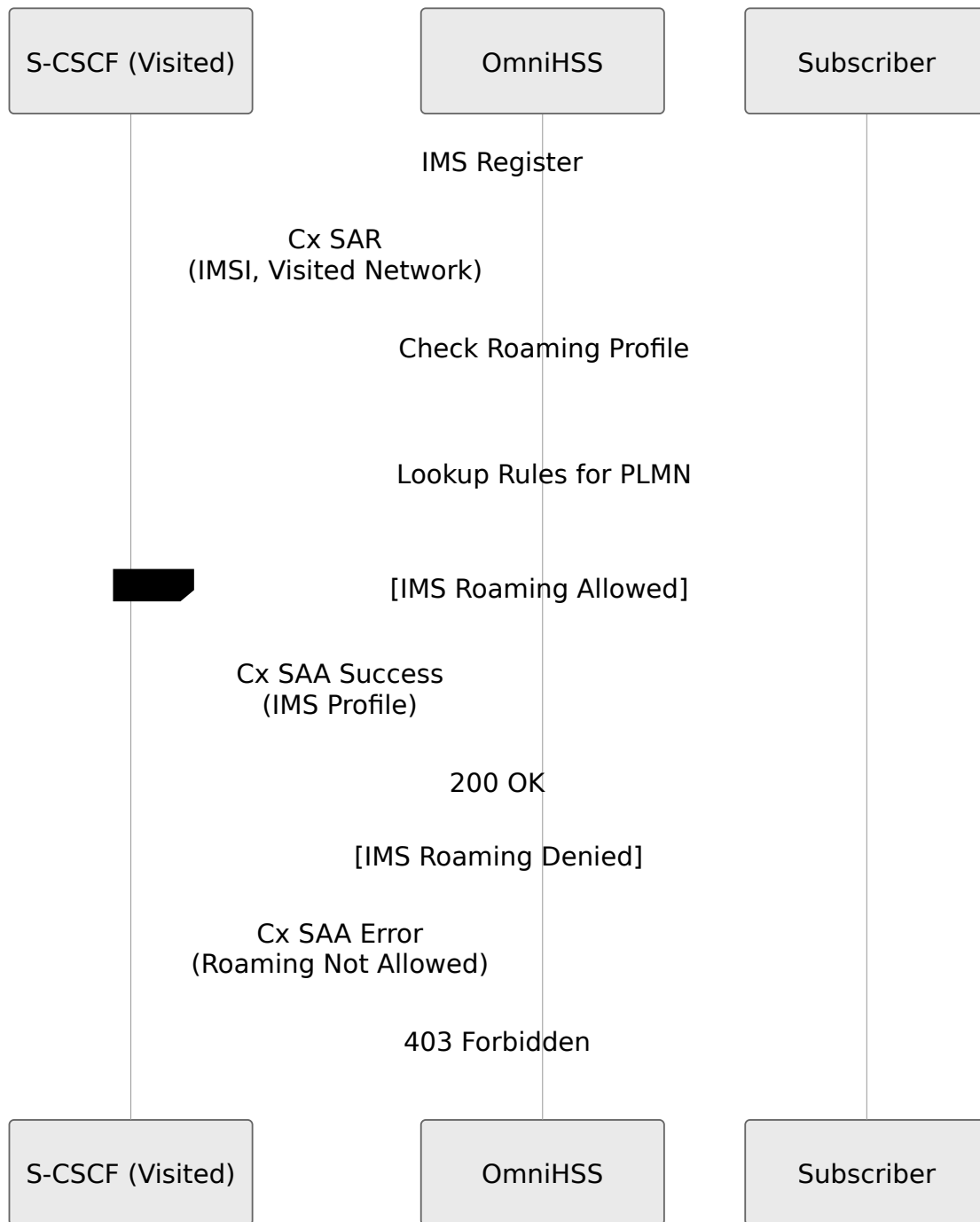
### S6a Interface (Data)

When subscriber attaches to visited network:



## Cx Interface (IMS)

When subscriber registers to IMS in visited network:



# Troubleshooting Roaming Issues

## Subscriber Cannot Attach in Visited Network

### Check roaming profile assignment:

- Query the database to view subscriber's assigned roaming profile

- Verify the profile name and default action settings

### **Check if rule exists for visited network:**

- Query the database for roaming rules matching the visited network's MCC/MNC
- Verify if a rule exists for the subscriber's roaming profile
- Check the data\_action value for that specific network

## **Subscriber Can Attach But Not Register IMS**

### **Check IMS action separately:**

- Query the roaming rules for the visited network
- Verify both data\_action and ims\_action values
- Look for cases where data is allowed but IMS is denied

## **Unexpected Roaming Behavior**

### **Review logs for roaming checks:**

```
[info] Roaming check: IMSI 001001123456789, Visited PLMN 310-410  
[info] Roaming rule matched: "Allow AT&T"  
[info] Data action: allow, IMS action: allow
```

---

# **Best Practices**

## **Profile Design**

1. **Start restrictive** - Default deny, explicitly allow partners
2. **Test thoroughly** - Verify rules in lab before production
3. **Document rules** - Maintain list of allowed networks and why
4. **Review regularly** - Update as roaming agreements change

# Rule Management

1. **Use descriptive names** - "Allow-ATT-Data-Only" not "Rule1"
2. **Verify MCC/MNC** - Double-check codes against official databases
3. **Consider both services** - Think about data and IMS separately
4. **Monitor usage** - Track which networks subscribers actually visit

# Operational Procedures

1. **Emergency Changes** - Have procedure to quickly enable/disable roaming
2. **Bulk Updates** - Plan for updating multiple subscribers' roaming profiles
3. **Reporting** - Track roaming usage and denied attempts
4. **Customer Communication** - Notify customers of roaming policy changes

---

[← Back to Operations Guide](#) | [Next: Protocol Flows →](#)

# Sh RepositoryData (Transparent Data)

OmniHSS acts as a **transparent repository** for service data that Application Servers (AS) store against a subscriber over the [Sh interface](#). Documents such as **MMTEL-Services** (supplementary services / call diversion, call waiting, OIP/OIR, barring) and **IMS-ODB-Information** (operator-determined barring) are written by the AS, kept by the HSS, and read back on demand.

"Transparent" has a precise meaning in [3GPP TS 29.328](#): the HSS understands this data **syntactically but not semantically**. It is opaque XML owned by the AS. OmniHSS stores and returns the `ServiceData` payload **byte-for-byte** — it never re-parses or re-serialises it — so a document an AS etags or checksums (for example XCAP/MMTel) is returned exactly as written.

The same documents can also be managed out-of-band by provisioning/CRM systems through the [REST Provisioning API](#).

## Table of Contents

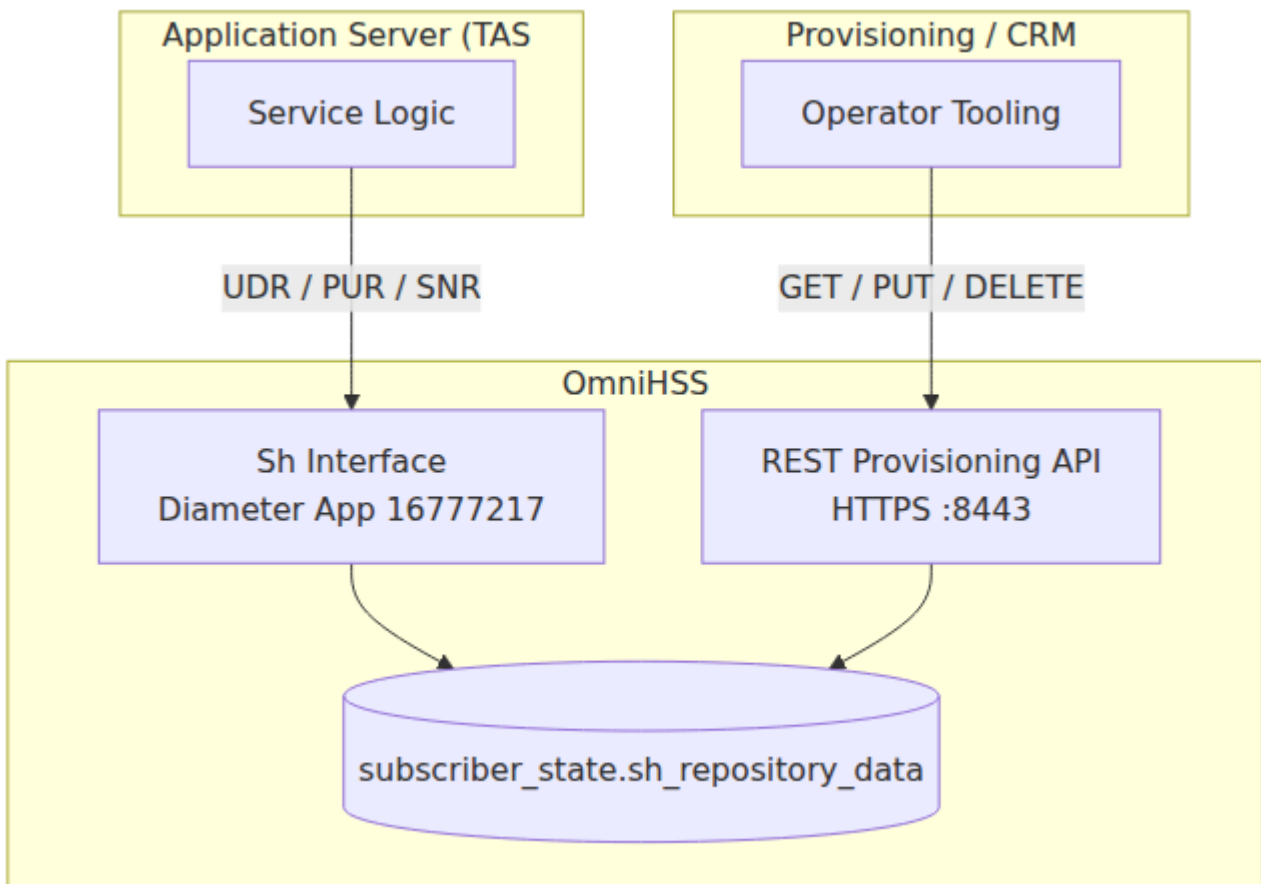
- [Concepts](#)
- [Data Model](#)
- [Sh Message Flows](#)
- [SequenceNumber and Concurrency](#)
- [REST Provisioning API](#)
- [Reference Tables](#)
- [Troubleshooting](#)

## Concepts

A subscriber may hold any number of independent **transparent documents**, each identified by a **Service-Indication** (the "filename"). A document has three parts:

Element	Owner	Meaning
<b>Service-Indication</b>	HSS-managed	Names the document (e.g. <code>MMTEL-Services</code> , <code>IMS-ODB-Information</code> ).
<b>SequenceNumber</b>	HSS-managed	Version counter used for optimistic concurrency. See <a href="#">SequenceNumber and Concurrency</a> .
<b>ServiceData</b>	AS-owned (opaque)	The XML payload itself. Stored and returned verbatim.

The HSS only interprets Service-Indication and SequenceNumber. The `ServiceData` body is treated as opaque bytes.



# Data Model

All of a subscriber's transparent documents are held in a single field, `subscriber_state.sh_repository_data`, as a JSON object keyed by Service-Indication. Each entry stores the version and the verbatim payload:

```
{
  "MMTEL-Services": { "sequence_number": "0", "service_data":
"<MMTelServices ...>...</MMTelServices>" },
  "IMS-ODB-Information": { "sequence_number": "1", "service_data":
"<OdbForImsOrientedServices xmlns=\"odb.mmtel.omnitou.ch\"/>" }
}
```

Field	Description
<code>sequence_number</code>	The current version of this document, as last accepted by the HSS. Returned in every <code>UDA</code> and used to validate updates.
<code>service_data</code>	The opaque XML body, stored exactly as the AS sent it (no namespace, attribute-order or whitespace normalisation). A document with no <code>ServiceData</code> element (the spec's <i>empty repository data</i> ) omits this field.

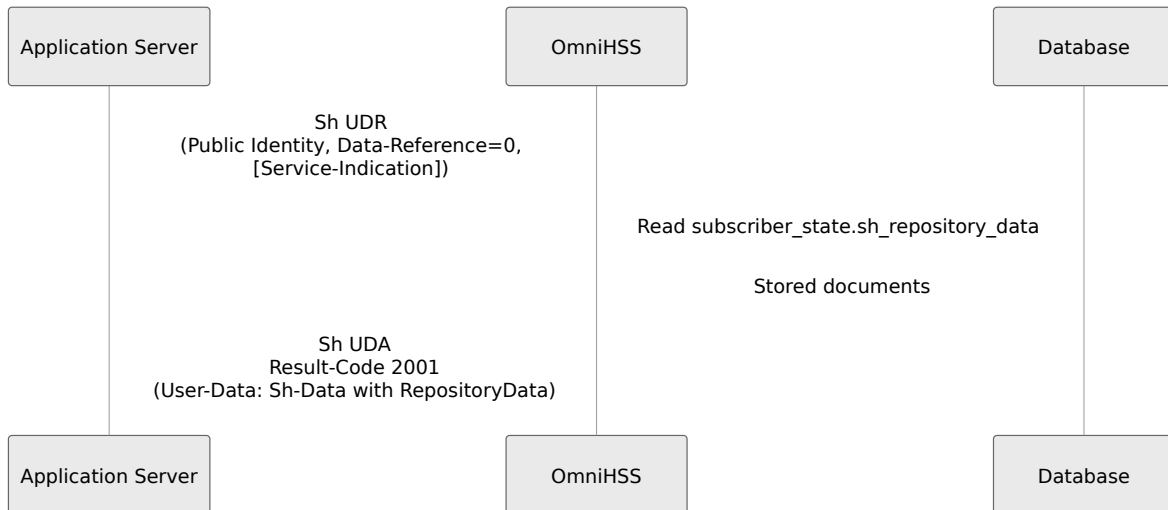
This model means a single subscriber can hold many distinct documents, each versioned independently, with no fixed schema imposed by the HSS.

## Sh Message Flows

RepositoryData is carried in the `User-Data` AVP as a `Sh-Data` XML document. RepositoryData is selected with **Data-Reference value 0**. An optional **Service-Indication** AVP narrows a request to a single document.

# User Data Request (UDR/UDA) – Read

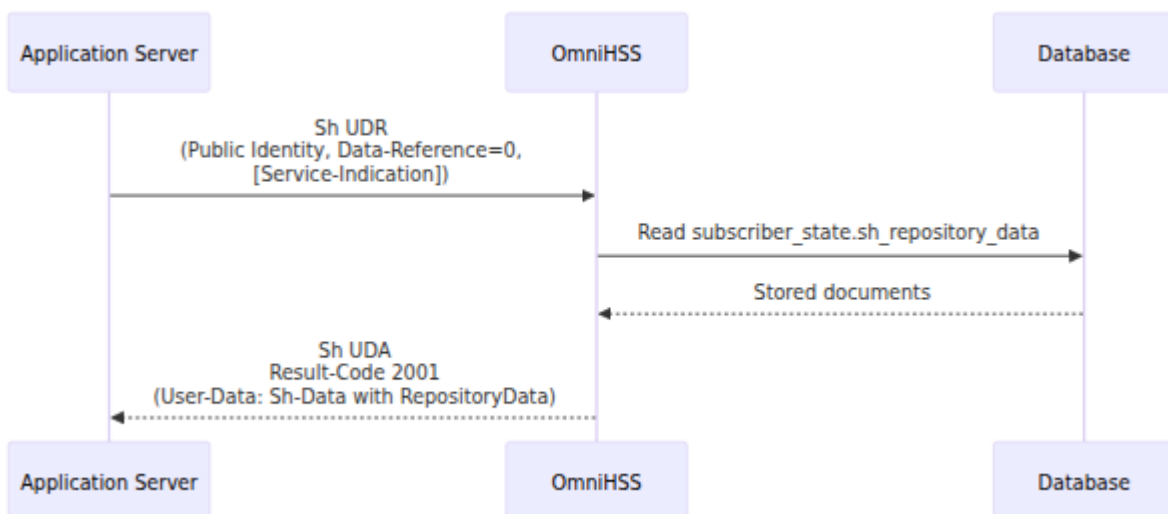
The AS reads one or all transparent documents. With a Service-Indication, only the matching document is returned; without one, every stored document is returned, each as its own `<RepositoryData>` element.



An unknown or non-IMS subscriber is answered with **Experimental-Result 5001 (DIAMETER\_ERROR\_USER\_UNKNOWN)**.

# Profile Update Request (PUR/PUA) – Write

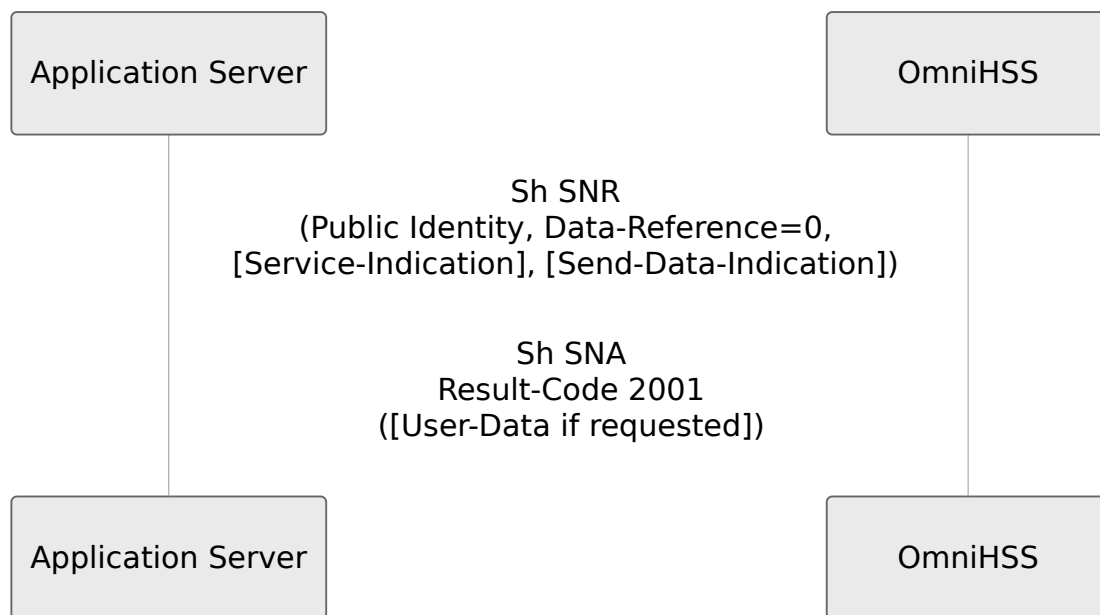
The AS creates or updates a single document. The `ServiceData` payload is stored verbatim. The update is accepted only if its SequenceNumber is valid (see below).



When a PUR resolves to multiple subscribers (a wildcarded Public Identity), the update is applied atomically: if any document is out of sync, none are persisted.

## Subscribe Notifications Request (SNR/SNA) — Subscribe

The AS subscribes to a subscriber's data and may request the current value be returned immediately (`Send-Data-Indication`). OmniHSS answers the SNR and, when requested, includes the current RepositoryData.



## SequenceNumber and Concurrency

The SequenceNumber provides optimistic concurrency control so two Application Servers cannot silently overwrite each other, per [3GPP TS 29.328](#).

- **Creating** a new document uses **SequenceNumber 0** (value `0` is reserved exclusively for newly-added data).
- **Updating** an existing document must carry the **immediate successor** of the stored SequenceNumber (stored + 1).
- The counter **wraps from 65535 back to 1** (not 0, since 0 is reserved).
- An update whose SequenceNumber is not the expected successor is rejected with **Experimental-Result 5105 (DIAMETER\_ERROR\_TRANSPARENT\_DATA\_OUT\_OF\_SYNC)** and the stored document is left unchanged.



Absent

PUR SequenceNumber  
0 (create)

Stored0

PUR SequenceNumber  
1 (stored+1)      DELETE (REST)

Stored1

PUR SequenceNumber  
n+1

StoredN

PUR with wrong  
SequenceNumber  
→ 5105, no change

**Example:** a document is stored at SequenceNumber 0. A PUR carrying SequenceNumber 5 is rejected with 5105 and nothing changes. A PUR carrying SequenceNumber 1 is accepted and the document advances to version 1.

## REST Provisioning API

The same documents can be read and written directly by provisioning/CRM systems. The subscriber is addressed by IMSI in the path; the document is identified by `service_indication` (in the request body for updates, or the

query string for reads/deletes). The opaque `service_data` is round-tripped byte-for-byte, exactly as over Sh.

All responses use the standard API envelope:

```
{ "status": "success", "response": { } }
```

Unlike the Sh path, the REST API is authoritative provisioning access and does **not** apply the SequenceNumber out-of-sync check — a write sets the document directly.

## List All Documents

Returns every stored document for the subscriber.

**Endpoint:** `GET /api/subscriber/repository_data/:imsi`

```
curl -k  
https://hss.example.com:8443/api/subscriber/repository_data/001001123
```

```
{  
  "status": "success",  
  "response": {  
    "MMTEL-Services": { "sequence_number": "0",  
"service_data": "<MMTelServices .../>" },  
    "IMS-ODB-Information": { "sequence_number": "1",  
"service_data": "<OdbForImsOrientedServices  
xmlns=\"odb.mmtel.omnitou.ch\"/>" }  
  }  
}
```

## Get a Single Document

Returns one document by Service-Indication.

**Endpoint:** `GET /api/subscriber/repository_data/:imsi?  
service_indication=:service_indication`

```
curl -k
"https://hss.example.com:8443/api/subscriber/repository_data/001001123
service_indication=MMTEL-Services"
```

```
{
  "status": "success",
  "response": { "sequence_number": "0", "service_data": "
<MMTelServices .../>" }
}
```

## Create or Update a Document

Creates the document if absent, or overwrites it.

**Endpoint:** `PUT /api/subscriber/repository_data/:imsi`

```
curl -k -X PUT
https://hss.example.com:8443/api/subscriber/repository_data/001001123
\
-H "Content-Type: application/json" \
-d '{
  "service_indication": "MMTEL-Services",
  "service_data": "<MMTelServices version=\"1\"/>",
  "sequence_number": "0"
}'
```

Field	Type	Required	Default	Description
<code>service_indication</code>	String	Yes	-	The document name to create/update (e.g. <code>MMTEL-Services</code> ).
<code>service_data</code>	String	No	-	The opaque XML payload, stored verbatim. Omit to store an empty document (Service-Indication and SequenceNumber only).
<code>sequence_number</code>	String	No	<code>0</code>	The version to record for the document.

The response echoes the stored document:

```
{
  "status": "success",
  "response": { "sequence_number": "0", "service_data": "<MMTelServices version=\"1\"/>" }
}
```

## Delete a Document

Removes a single document by Service-Indication.

**Endpoint:** `DELETE /api/subscriber/repository_data/:imsi`

```
curl -k -X DELETE
https://hss.example.com:8443/api/subscriber/repository_data/001001123
\
-H "Content-Type: application/json" \
-d '{ "service_indication": "MMTEL-Services" }'
```

Field	Type	Required	Default	Description
service_indication	String	Yes	-	The document to remove.

The response returns the remaining documents for the subscriber.

## Reference Tables

### Data-Reference Values (RepositoryData)

Value	Name	Description	Reference
0	RepositoryData	Transparent service data identified by Service-Indication	3GPP TS 29.328 §7.6

## Result and Experimental-Result Codes

Code	Name	Meaning
2001	DIAMETER_SUCCESS	UDR/PUR/SI processed successfully
5001	DIAMETER_ERROR_USER_UNKNOWN	Subscriber not found or not enabled
5105	DIAMETER_ERROR_TRANSPARENT_DATA_OUT_OF_SYNC	Update SequenceN is not the successor of stored value

## Application ID

ID	Interface	Description	Reference
16777217	Sh	AS ↔ HSS user data	<a href="#">3GPP TS 29.329</a>

# Common Service-Indications

Service-Indication	Typical Content
MMTEL-Services	MMTel supplementary services (call diversion, communication waiting/hold/barring, OIP/OIR, conferencing) — simservs XML
IMS-ODB-Information	Operator-determined barring information for IMS-oriented services

Service-Indication values are defined by the Application Server and its corresponding XML schema; the HSS imposes no fixed list.

## Troubleshooting

### Application Server receives 5105 on update

**Symptoms:** A PUR is answered with Experimental-Result 5105 and the document is unchanged.

**Possible causes:**

- The AS sent a SequenceNumber that is not the immediate successor of the value stored at the HSS (its cached copy is stale).
- A concurrent update from another AS advanced the SequenceNumber.

**Resolution:**

1. Read the current document with a UDR to obtain the up-to-date SequenceNumber.
2. Re-issue the PUR with SequenceNumber = stored + 1.
3. To force a value administratively, set it via the REST API, which does not apply the concurrency check.

# Returned ServiceData differs from what was written

**Symptoms:** The AS believes the payload was altered by the HSS.

## Possible causes:

- This should not occur — OmniHSS stores and returns `ServiceData` byte-for-byte. Apparent differences usually come from the AS's own XML library re-serialising the document (changing namespace prefixes, attribute order or whitespace) before comparison.

## Resolution:

1. Compare the raw bytes returned in the `UDA/REST` response against the raw bytes sent, not a re-parsed representation.
2. Confirm the AS is not normalising the XML on its side before the comparison.

# Document not found via REST

**Symptoms:** `GET .../repository_data/:imsi?service_indication=...` returns an error.

## Possible causes:

- No document with that Service-Indication is stored for the subscriber.
- The IMSI does not correspond to an existing subscriber.

## Resolution:

1. List all documents with `GET /api/subscriber/repository_data/:imsi` to see what is stored.
2. Verify the IMSI with [Get Subscriber by IMSI](#).

# OmniHSS

# Troubleshooting Guide

[← Back to Operations Guide](#)

---

## Table of Contents

- [Troubleshooting Overview](#)
  - [Authentication Failures](#)
  - [Diameter Connectivity Issues](#)
  - [Database Issues](#)
  - [EPC Registration Failures](#)
  - [IMS Registration Failures](#)
  - [VoLTE Call Failures](#)
  - [Roaming Issues](#)
  - [EIR Problems](#)
  - [Performance Problems](#)
  - [Subscriber State Issues](#)
  - [API Issues](#)
  - [Diagnostic Tools and Commands](#)
-

# **Troubleshooting Overview**

## **General Troubleshooting Approach**

Issue Reported

OmniCore  
5GC ▼

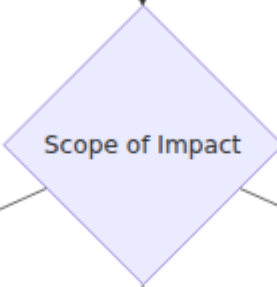
OmniCall  
▼

OmniRAN  
▼

OmniCharge  
▼

Platform  
▼

English ▼



Single Subscriber

Check Subscriber Data

Verify Subscriber Config

Multiple Subscribers

Check System/Profile

Verify Profile Config

All Subscribers

Check System Health

Check Diameter/DB/System

Review Logs

Identify Root Cause

Apply Fix

Verify Resolution

Document Solution

# Information to Gather

Before troubleshooting any issue, collect:

## 1. **Subscriber Information** (if subscriber-specific)

- IMSI
- MSISDN (phone number)
- Last known state
- Error messages from device

## 2. **Timing Information**

- When did the issue start?
- Is it intermittent or constant?
- Time of last successful operation

## 3. **Scope of Impact**

- Single subscriber or multiple?
- Specific network or all networks?
- Specific service (data/voice) or both?

## 4. **System State**

- Check **Control Panel** for system status
  - Review Diameter peer status
  - Verify database connectivity
- 

# Authentication Failures

## Symptoms

- Subscriber cannot attach to network
- "Authentication rejected" errors
- Repeated authentication attempts

# Common Causes and Solutions

## Cause 1: Incorrect Key Set

### Symptoms:

- Consistent authentication failure for specific subscriber
- Works for other subscribers with same profile

### Diagnostic Steps:

1. Query subscriber to verify key\_set\_id:

```
curl -k https://hss.example.com:8443/api/subscriber/imsi/[IMSI]
```

2. Verify key set exists and has correct values:

```
curl -k https://hss.example.com:8443/api/key_set/[KEY_SET_ID]
```

3. Compare Ki and OPC values with SIM card documentation

### Solution:

- Update subscriber with correct **key set**
- If keys are correct, SIM card may be faulty

## Cause 2: SQN Out of Sync

### Symptoms:

- Authentication fails after previously working
- Error: "SQN synchronization failure"
- Works intermittently

### Diagnostic Steps:

1. Check subscriber state for SQN value in database
2. Look for SQN-related errors in logs

### 3. Verify subscriber's key set SQN value

#### **Solution:**

- SQN will automatically resynchronize after subscriber sends AUTS
- If persistent, reset SQN to 0 in key set (requires subscriber re-attach)

**Warning:** Resetting SQN can cause security issues. Only do during maintenance.

### **Cause 3: Subscriber Disabled**

#### **Symptoms:**

- Authentication rejected immediately
- No authentication vectors generated

#### **Diagnostic Steps:**

##### 1. Check subscriber enabled status:

```
curl -k https://hss.example.com:8443/api/subscriber/imsi/[IMSI]
```

##### 2. Verify `enabled` field is `true`

#### **Solution:**

- **Enable subscriber:**

```
curl -k -X PUT https://hss.example.com:8443/api/subscriber/[ID] \
-H "Content-Type: application/json" \
-d '{"subscriber": {"enabled": true}}'
```

### **Cause 4: Missing EPC Profile**

#### **Symptoms:**

- Subscriber lookup succeeds but authentication fails
- Error: "No EPC profile assigned"

## Diagnostic Steps:

1. Check subscriber's `epc_profile_id` field
2. Verify EPC profile exists:

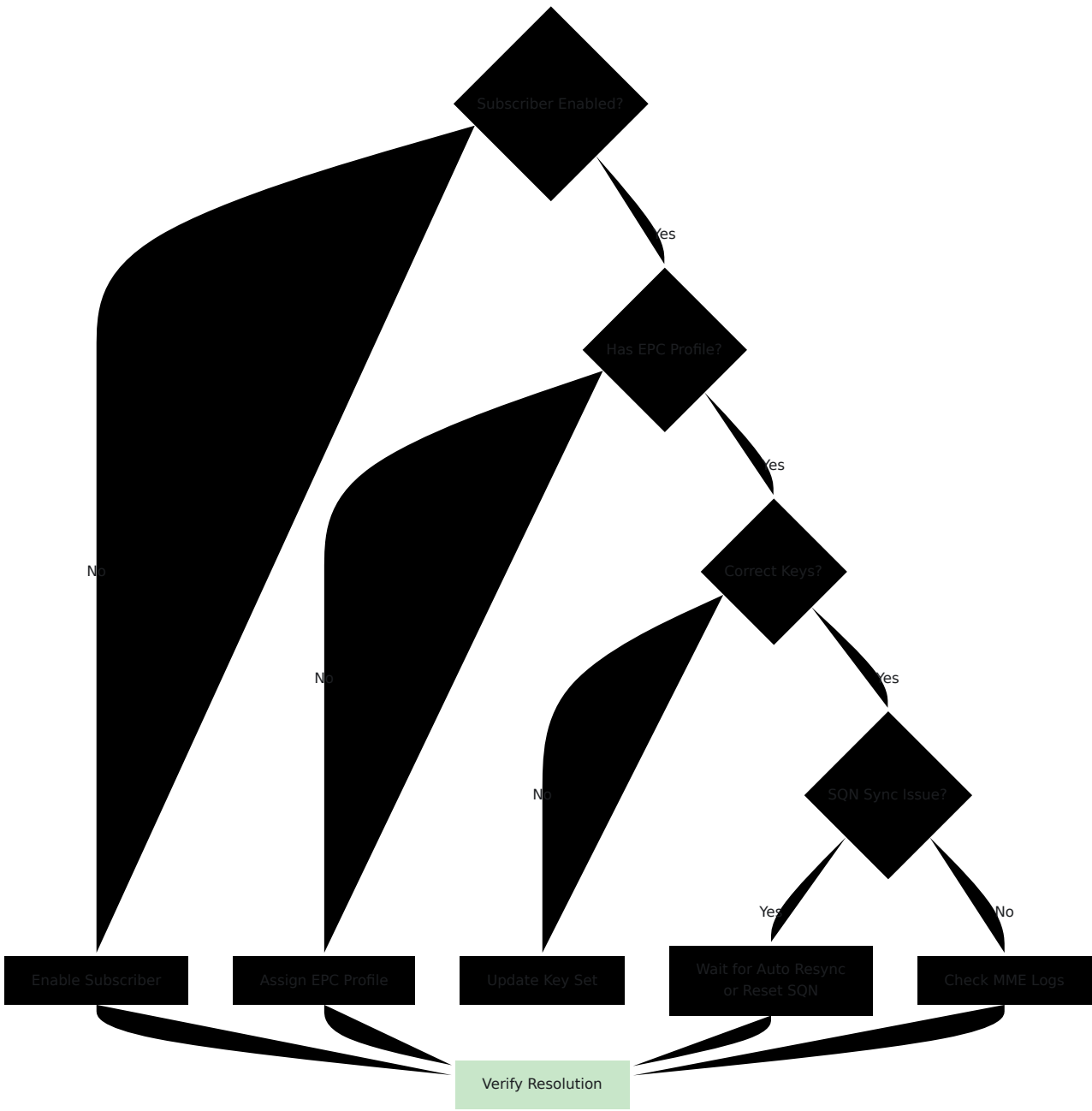
```
curl -k  
https://hss.example.com:8443/api/epc/profile/[PROFILE_ID]
```

## Solution:

- Assign valid **EPC profile** to subscriber

# Authentication Troubleshooting Flowchart

Authentication Failure



# Diameter Connectivity Issues

## Symptoms

- Diameter peers showing as disconnected in [Control Panel](#)
- "No route to host" errors
- Services failing for all subscribers

## Common Causes and Solutions

### Cause 1: Network Connectivity

#### Symptoms:

- Peer never connects
- Connection timeout errors
- Ping fails to peer

#### Diagnostic Steps:

1. Verify network connectivity from OmniHSS to peer:

```
ping [PEER_IP]
```

2. Check if Diameter port is reachable:

```
telnet [PEER_IP] 3868
```

3. Verify firewall rules allow Diameter traffic (port 3868)

#### Solution:

- Fix network routing
- Update firewall rules
- Verify peer is running and listening

## Cause 2: Incorrect Diameter Configuration

### Symptoms:

- Connection attempts fail
- CER/CEA exchange fails
- Peer rejects connection

### Diagnostic Steps:

1. Review runtime.exs Diameter configuration:
  - Verify peer origin\_host matches peer's expected value
  - Check origin\_realm configuration
  - Verify peer IP address is correct
2. Check logs for CER/CEA errors
3. Verify peer's configuration expects OmniHSS's origin\_host

### Solution:

- Update runtime.exs with correct **Diameter configuration**
- Restart OmniHSS after configuration change
- Coordinate with peer administrator to verify settings

## Cause 3: Certificate Issues (TLS Diameter)

### Symptoms:

- Connection fails during TLS handshake
- Certificate validation errors
- "Certificate expired" or "Certificate invalid" errors

### Diagnostic Steps:

1. Verify certificate files exist in `priv/cert/`
2. Check certificate expiration:

```
openssl x509 -in priv/cert/diameter.crt -noout -dates
```

3. Verify certificate chain is complete
4. Check peer's certificate if mutual TLS

### **Solution:**

- Renew expired certificates
- Install correct certificate chain
- Update certificate files and restart OmniHSS

### **Cause 4: Peer Application Support Mismatch**

#### **Symptoms:**

- Peer connects but doesn't support required applications
- Capabilities exchange succeeds but operations fail
- "Application not supported" errors

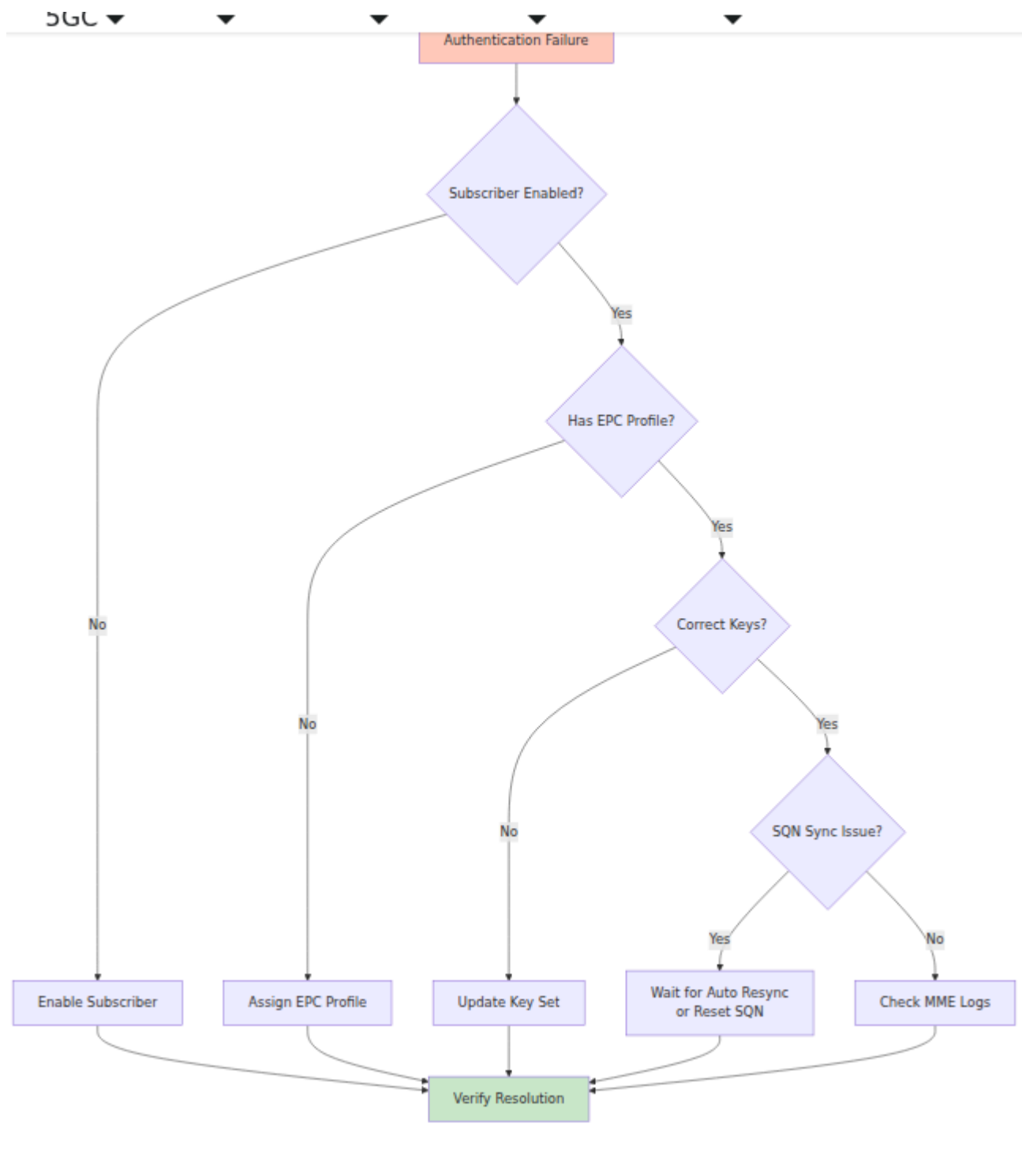
#### **Diagnostic Steps:**

1. Check [Control Panel Diameter page](#) for peer applications
2. Verify peer supports required application (S6a, Cx, Sh, etc.)
3. Review CER/CEA exchange in logs

#### **Solution:**

- Verify peer configuration includes required Diameter applications
- Check that peer type matches expected functionality:
  - MME must support S6a (16777251)
  - S-CSCF must support Cx (16777216)
  - P-GW must support Gx (16777238)

# Diameter Troubleshooting Flowchart



# Database Issues

## Symptoms

- API returns 500 errors
- Control Panel fails to load
- "Database connection failed" errors
- Slow query performance

## Common Causes and Solutions

### Cause 1: Database Server Down

#### Symptoms:

- All API calls fail
- Control Panel shows error
- "Connection refused" errors

#### Diagnostic Steps:

1. Test database connectivity:

```
# Connect to the database  
psql -h [DB_HOST] -U [DB_USER] -d [DB_NAME]
```

2. Check database service status on database server
3. Verify network connectivity to database server

#### Solution:

- Start database service
- Fix database server issues
- Verify network routing to database server

## **Cause 2: Incorrect Database Credentials**

### **Symptoms:**

- "Authentication failed" errors
- OmniHSS can't connect at startup

### **Diagnostic Steps:**

1. Review database configuration in runtime.exs
2. Test credentials manually with database client
3. Check database user permissions

### **Solution:**

- Update **database configuration** in runtime.exs
- Grant correct permissions to database user
- Restart OmniHSS after configuration change

## **Cause 3: Connection Pool Exhausted**

### **Symptoms:**

- Intermittent 500 errors
- "No available connections" errors
- High load periods trigger failures

### **Diagnostic Steps:**

1. Check current connection count in database
2. Review database pool size in runtime.exs
3. Monitor connection usage during peak load

### **Solution:**

- Increase pool size in runtime.exs configuration
- Investigate connection leaks if pool exhausts repeatedly
- Consider database scaling if load is consistently high

## **Cause 4: Slow Queries**

### **Symptoms:**

- API responses very slow
- Timeouts on subscriber lookups
- Database CPU high

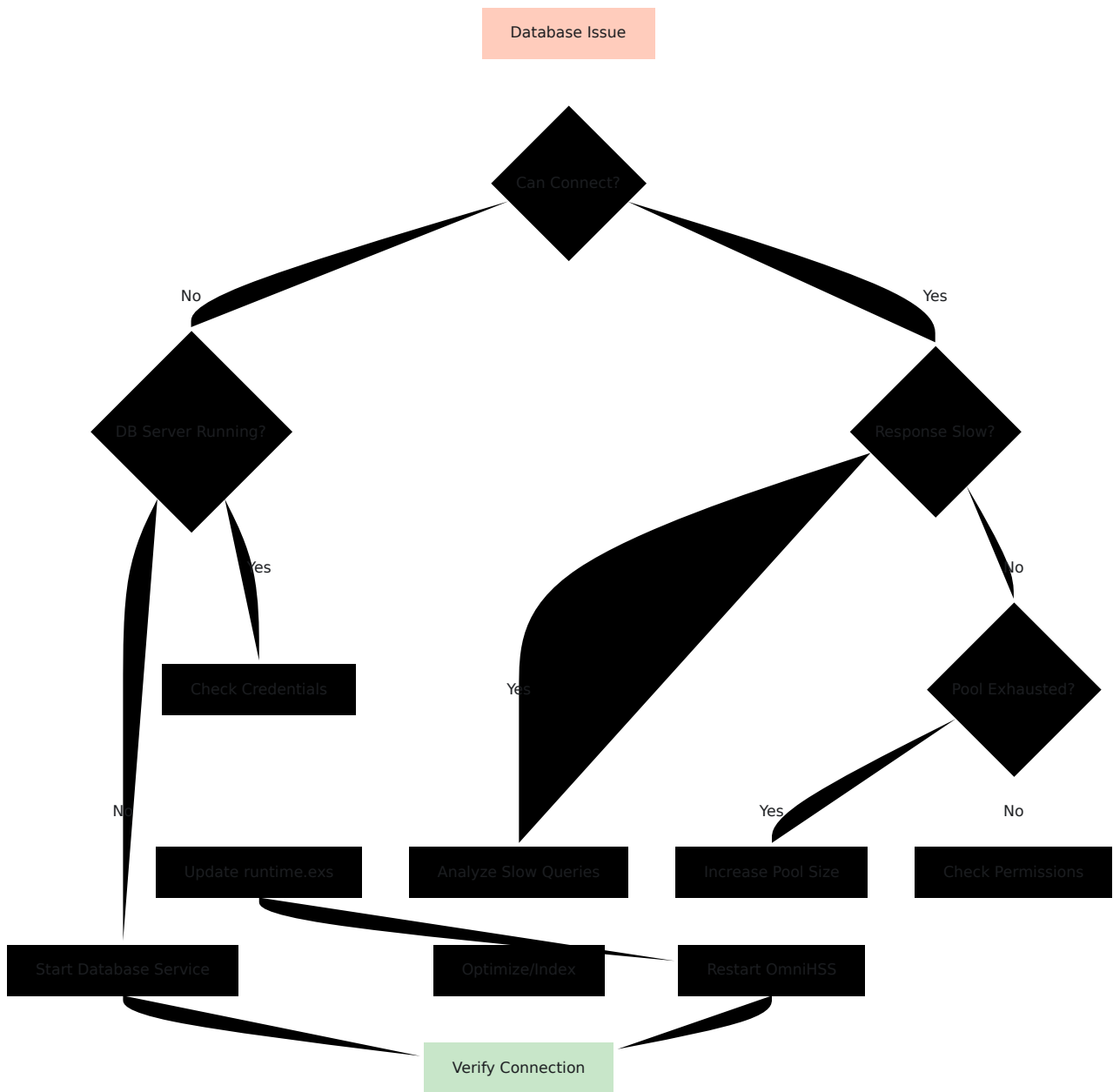
### **Diagnostic Steps:**

1. Query database for slow query log
2. Identify specific slow queries
3. Check for missing indexes
4. Verify subscriber count and table sizes

### **Solution:**

- Optimize slow queries
- Add missing indexes
- Consider database performance tuning
- Plan for database scaling if needed

# Database Troubleshooting Flowchart



## EPC Registration Failures

### Symptoms

- Subscriber cannot attach to LTE network
- MME rejects attachment
- No PDN session established

# Common Causes and Solutions

## Cause 1: Roaming Denied

### Symptoms:

- Subscriber works on home network but fails when roaming
- "Roaming not allowed" errors
- Works for some networks but not others

### Diagnostic Steps:

1. Check subscriber's roaming\_profile\_id
2. Query roaming profile and rules
3. Verify MCC/MNC of visited network
4. Check if roaming rule exists for that network

### Solution:

- Add [roaming rule](#) for visited network MCC/MNC
- Or update roaming profile default action to allow
- See [Roaming Documentation](#) for configuration

## Cause 2: Missing APN Configuration

### Symptoms:

- Attachment succeeds but PDN session fails
- "Unknown APN" errors from MME
- Subscriber can't get data connection

### Diagnostic Steps:

1. Check EPC profile has APN profiles linked
2. Verify APN identifier matches what device requests
3. Query APN profile configuration

### Solution:

- Link [APN profiles](#) to subscriber's EPC profile
- Ensure APN name matches device configuration
- Verify APN QoS profile exists

### **Cause 3: MME Not Connected**

#### **Symptoms:**

- All subscribers fail to attach
- No communication with MME
- Diameter peer down

#### **Diagnostic Steps:**

1. Check [Control Panel Diameter page](#)
2. Verify MME peer status is "Connected"
3. Check MME supports S6a application

#### **Solution:**

- Troubleshoot [Diameter connectivity](#)
- Verify MME configuration
- Contact MME administrator

### **Cause 4: Subscriber State Corruption**

#### **Symptoms:**

- Subscriber shows as attached but can't attach again
- State doesn't match reality
- Detach and re-attach fails

#### **Diagnostic Steps:**

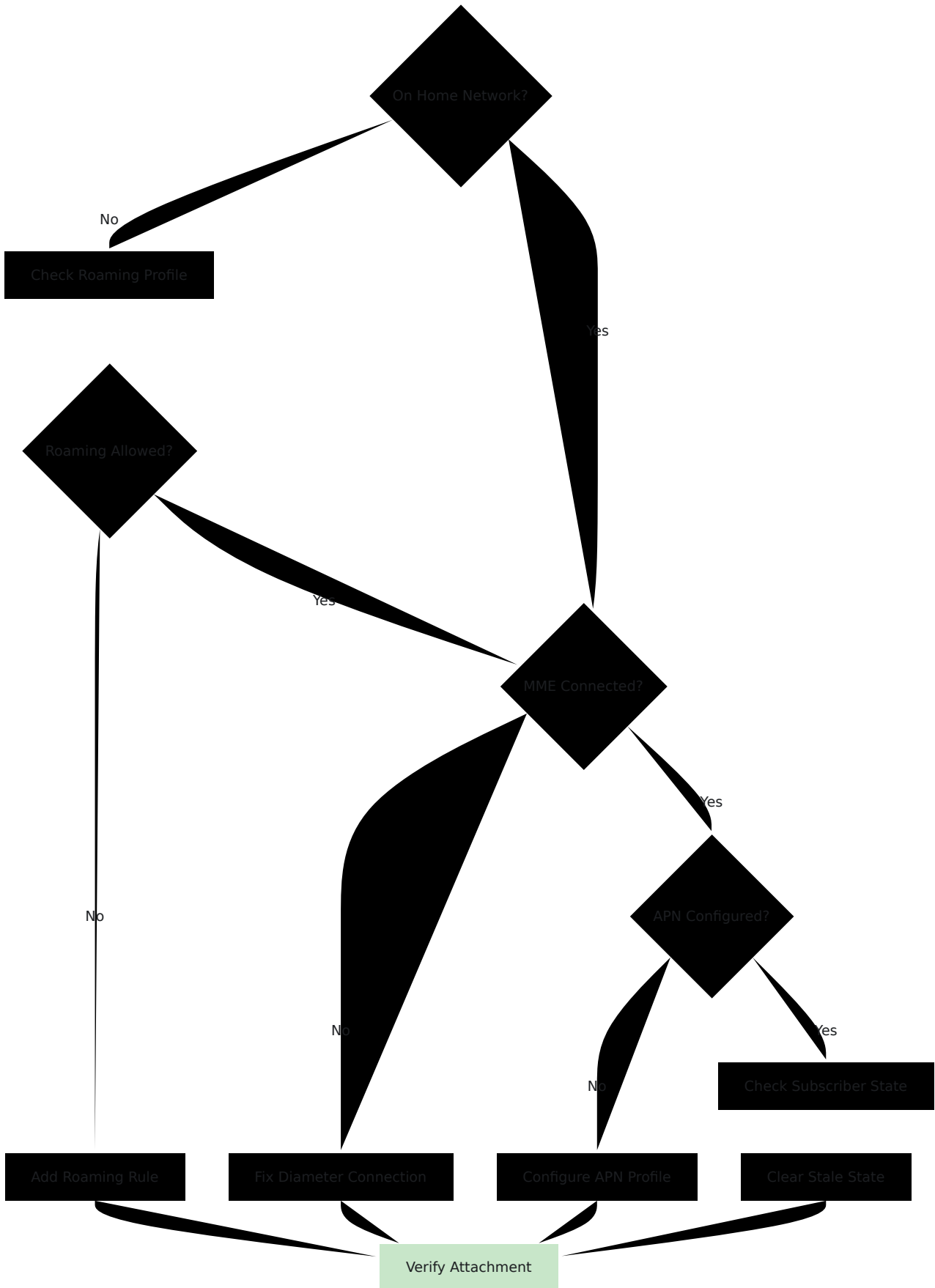
1. Query subscriber state from database
2. Check for stale MME assignments
3. Verify last update timestamp

#### **Solution:**

- Clear subscriber state (detach procedure)
- Reset serving MME in subscriber state
- May require subscriber power cycle

# EPC Registration Troubleshooting Flowchart

EPC Registration Fails



---

# IMS Registration Failures

## Symptoms

- Subscriber can't register for VoLTE
- "IMS registration failed" on device
- Data works but voice doesn't

## Common Causes and Solutions

### Cause 1: IMS Disabled for Subscriber

#### Symptoms:

- Subscriber has data but no IMS
- Registration rejected immediately

#### Diagnostic Steps:

1. Query subscriber and check `ims_enabled` field
2. Verify subscriber has `ims_profile_id` assigned

#### Solution:

- [Enable IMS](#) for subscriber
- Assign [IMS profile](#)

### Cause 2: S-CSCF Not Connected

#### Symptoms:

- All IMS registrations fail
- No IMS-related Diameter traffic

#### Diagnostic Steps:

1. Check [Control Panel Diameter page](#)

2. Verify S-CSCF peer connected
3. Check S-CSCF supports Cx application

**Solution:**

- Fix [Diameter connectivity](#) to S-CSCF
- Verify S-CSCF configuration

**Cause 3: Missing or Invalid IFC Template****Symptoms:**

- Registration fails during User-Authorization-Answer
- IFC-related errors in logs

**Diagnostic Steps:**

1. Query subscriber's IMS profile
2. Verify IFC template is present
3. Check IFC XML syntax

**Solution:**

- Update [IMS profile](#) with valid IFC template
- See [Profiles Documentation](#) for IFC examples

**Cause 4: Roaming Denied for IMS****Symptoms:**

- IMS works on home network
- Fails when roaming
- Data roaming works but not IMS

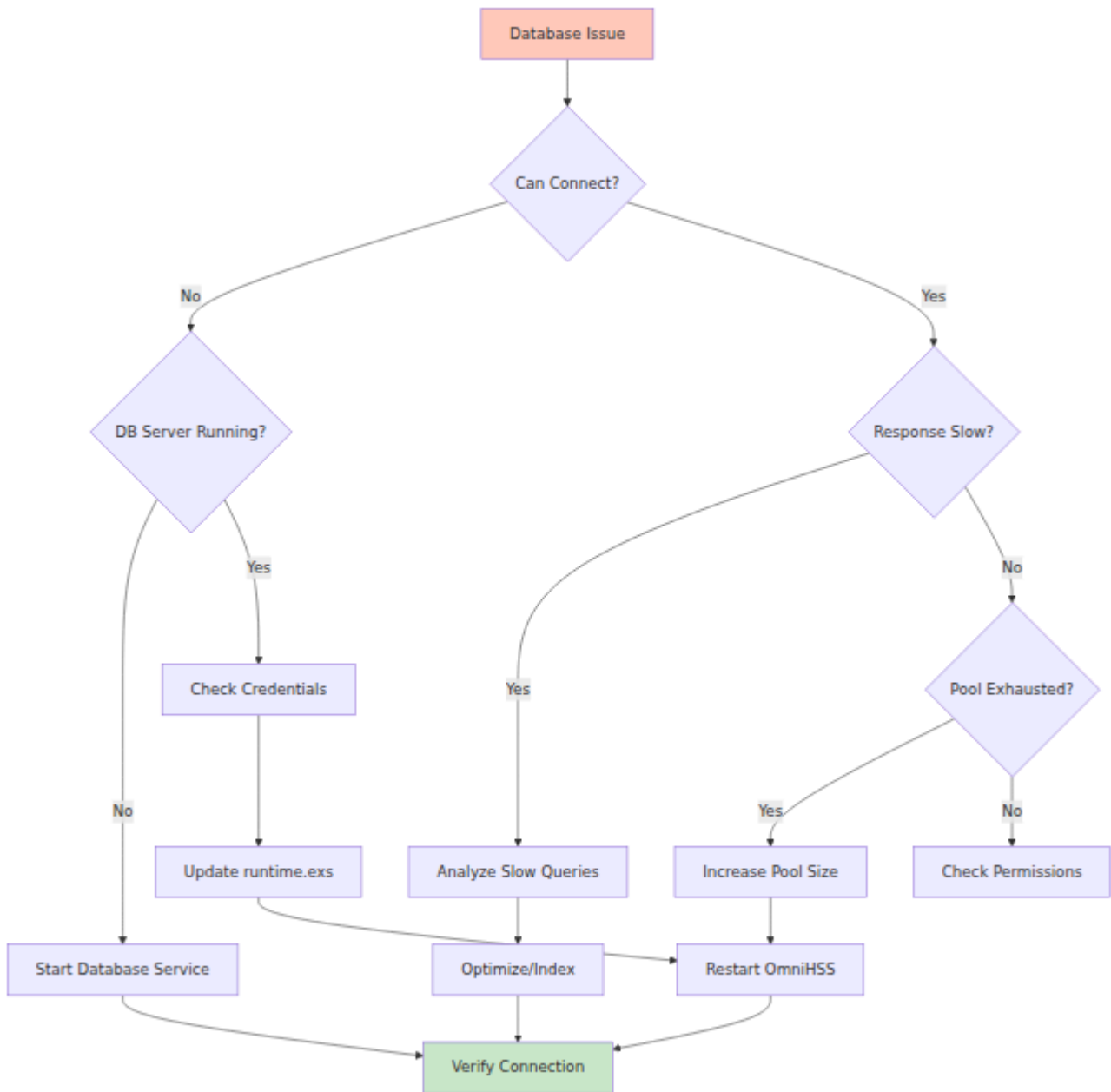
**Diagnostic Steps:**

1. Check roaming profile IMS action
2. Verify roaming rules have correct `ims_action`

**Solution:**

- Update **roaming rules** to allow IMS
- Or update roaming profile default IMS action

## IMS Registration Troubleshooting Flowchart



# VoLTE Call Failures

## Symptoms

- IMS registration succeeds but calls fail
- One-way audio
- Call drops immediately
- "Call failed" error on device

## Common Causes and Solutions

### Cause 1: P-CSCF Not Connected

#### Symptoms:

- Registration works but calls fail
- Media authorization fails

#### Diagnostic Steps:

1. Check [Control Panel Diameter page](#)
2. Verify P-CSCF peer connected
3. Check P-CSCF supports Rx application (OmniHSS PCRF function)

#### Solution:

- Fix [Diameter connectivity](#) to P-CSCF
- Verify P-CSCF configuration points to OmniHSS for Rx

### Cause 2: Missing Media Authorization

#### Symptoms:

- Call setup starts but fails
- AAR/AAA exchange fails
- Rx interface errors

#### Diagnostic Steps:

1. Check logs for Rx Diameter messages
2. Verify AAR (AA-Request) received
3. Check AAA (AA-Answer) response

**Solution:**

- Verify P-CSCF is sending AAR for media authorization
- Check OmniHSS Rx application configuration
- Verify subscriber has active IMS registration

**Cause 3: QoS/Bearer Issues****Symptoms:**

- Call connects but no audio
- One-way audio
- Quality issues

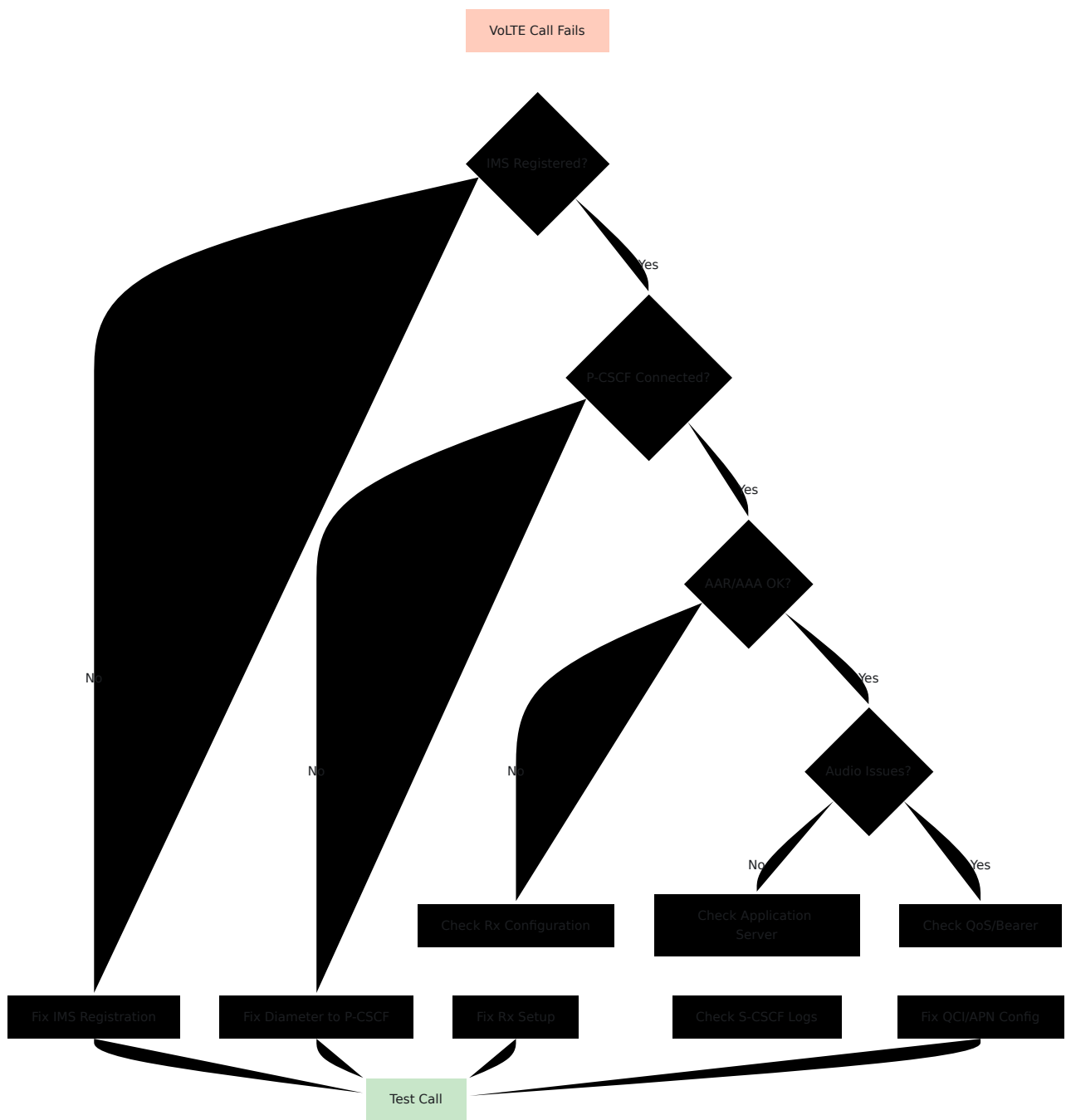
**Diagnostic Steps:**

1. Check APN QoS profile for voice APN
2. Verify QCI is set correctly (typically QCI 1 for voice)
3. Check P-GW is connected for Gx (PCRF function)

**Solution:**

- Verify **APN QoS profile** for IMS APN
- Ensure QCI 1 is configured for voice bearer
- Fix **Diameter connectivity** to P-GW if needed

# VoLTE Call Troubleshooting Flowchart



## Roaming Issues

### Symptoms

- Subscriber works at home but not when roaming

- Some roaming networks work, others don't
- Roaming data works but not voice (or vice versa)

## Common Causes and Solutions

### Cause 1: No Roaming Profile Assigned

#### Symptoms:

- Roaming fails for subscriber
- Other subscribers roam successfully

#### Diagnostic Steps:

1. Query subscriber's `roaming_profile_id`
2. Check if field is null

#### Solution:

- Assign **roaming profile** to subscriber

### Cause 2: Roaming Denied by Policy

#### Symptoms:

- Roaming fails consistently on specific network
- Error indicates policy rejection

#### Diagnostic Steps:

1. Identify visited network MCC/MNC from subscriber device or MME
2. Query subscriber's roaming profile
3. Check roaming rules for matching MCC/MNC
4. Check profile's default action

#### Solution:

- Add **roaming rule** to allow visited network:

```
curl -k -X POST https://hss.example.com:8443/api/roaming/rule \
-H "Content-Type: application/json" \
-d '{
  "roaming_rule": {
    "name": "Allow Visited Network",
    "mcc": "310",
    "mnc": "410",
    "data_action": "allow",
    "ims_action": "allow"
  }
}'
```

### Cause 3: Data Allowed but IMS Denied

#### Symptoms:

- Data roaming works
- Voice/IMS roaming fails
- Split service availability

#### Diagnostic Steps:

1. Query roaming rules for visited network
2. Check `data_action` vs `ims_action` values
3. Verify roaming profile default actions

#### Solution:

- Update roaming rule to allow IMS:
  - Set `ims_action: "allow"`
- Or update profile's `ims_action_if_no_rules_match` to `"allow"`

See [Roaming Documentation](#) for detailed configuration.

---

# EIR Problems

## Symptoms

- Devices blocked unexpectedly
- Stolen devices not blocked
- EIR check failing

## Common Causes and Solutions

### Cause 1: Incorrect IMEI Regex

#### Symptoms:

- Wrong devices blocked/allowed
- Rule matches incorrectly

#### Diagnostic Steps:

1. Query EIR rules
2. Identify which rule is matching
3. Test regex pattern against actual IMEI
4. Check rule priority/order

#### Solution:

- Update **EIR rule** with correct regex
- Test regex thoroughly before applying
- Consider rule order (first match wins)

### Cause 2: MME Not Sending S13 Requests

#### Symptoms:

- EIR check never happens
- All devices allowed regardless of rules

#### Diagnostic Steps:

1. Check if MME is configured to use S13 interface
2. Verify MME Diameter peer connected
3. Check for S13 application support
4. Review MME configuration

**Solution:**

- Configure MME to perform EIR checks via S13
- Verify Diameter peer supports S13 application (16777252)
- Contact MME administrator if needed

**Cause 3: No Default Rule****Symptoms:**

- Devices not matching any rule have unexpected behavior

**Diagnostic Steps:**

1. Query all EIR rules
2. Check if catch-all rule exists
3. Verify rule ordering

**Solution:**

- Add default rule with regex `.*` to match all IMEIs
  - Set appropriate action (whitelist or blacklist)
  - Ensure specific rules are checked before catch-all
- 

# Performance Problems

## Symptoms

- Slow API responses
- Diameter request timeouts
- High CPU or memory usage

- Control Panel slow to load

## Common Causes and Solutions

### Cause 1: High Database Load

#### Symptoms:

- All operations slow
- Database CPU high
- Query timeouts

#### Diagnostic Steps:

1. Check database server resource usage
2. Identify slow queries
3. Check for missing indexes
4. Monitor query patterns

#### Solution:

- Optimize slow queries
- Add database indexes
- Increase database resources
- Consider database scaling
- See [Database Issues](#)

### Cause 2: High Subscriber Count

#### Symptoms:

- Performance degraded over time
- Slowness correlates with subscriber growth
- List operations especially slow

#### Diagnostic Steps:

1. Query total subscriber count

2. Check table sizes
3. Review query execution plans
4. Monitor resource usage trends

**Solution:**

- Plan capacity upgrade
- Optimize queries for large datasets
- Consider pagination for large results
- Implement caching if needed

**Cause 3: Diameter Peer Issues****Symptoms:**

- Diameter operations slow
- Timeouts on specific peer
- Some peers fast, others slow

**Diagnostic Steps:**

1. Check [Control Panel Diameter page](#)
2. Identify slow peer
3. Test network latency to peer
4. Check peer resource usage

**Solution:**

- Investigate peer performance issues
- Check network path for congestion
- Consider adding redundant peers
- Increase Diameter timeout if needed

**Cause 4: Memory Issues****Symptoms:**

- OmniHSS memory usage high

- Out of memory errors
- Performance degrades over time

### **Diagnostic Steps:**

1. Check OmniHSS memory usage on Application page
2. Monitor memory trend
3. Check for memory leaks
4. Review Erlang VM settings

### **Solution:**

- Restart OmniHSS to clear temporary condition
  - Investigate memory leak if usage continuously grows
  - Adjust Erlang VM memory settings in runtime.exs
  - Plan for hardware upgrade if consistently high
- 

# **Subscriber State Issues**

## **Symptoms**

- Subscriber shows as attached but isn't
- Stale state information
- Location information incorrect
- Can't detach subscriber

## **Common Causes and Solutions**

### **Cause 1: MME Crash/Restart**

#### **Symptoms:**

- Subscriber shows serving MME that is no longer serving
- Subscriber can't attach after MME restart
- State is stale

### **Diagnostic Steps:**

1. Check subscriber state for serving MME
2. Verify if MME has restarted
3. Check MME's last connection time

### **Solution:**

- Wait for subscriber to attach again (state will update)
- Or manually clear subscriber state
- MME should send Cancel-Location on restart

### **Cause 2: Network Detach Not Received**

#### **Symptoms:**

- Subscriber powered off but shows as attached
- PDN sessions remain in database
- Location not cleared

#### **Diagnostic Steps:**

1. Check subscriber's last\_seen timestamp
2. Verify if old state (hours or days old)
3. Check if subscriber device is reachable

#### **Solution:**

- State will clear when subscriber attaches again
- Or wait for state timeout (if implemented)
- Manual cleanup may be required for very stale state

### **Cause 3: Database Corruption**

#### **Symptoms:**

- Inconsistent state across tables
- Foreign key violations
- State doesn't make sense

## **Diagnostic Steps:**

1. Query subscriber state directly from database
2. Check for orphaned records
3. Verify referential integrity

## **Solution:**

- Identify and fix inconsistent data
  - May require manual database cleanup
  - Contact support if corruption is widespread
- 

# **API Issues**

## **Symptoms**

- API returns errors
- Slow API responses
- Cannot create/update entities
- 500 errors

## **Common Causes and Solutions**

### **Cause 1: Invalid Request Data**

#### **Symptoms:**

- 400 or 422 errors
- Validation error messages
- Field rejected

#### **Diagnostic Steps:**

1. Review error response for specific field errors
2. Check API request format

3. Verify required fields present
4. Check data types

### **Solution:**

- Fix request data to match [API reference](#)
- Ensure all required fields included
- Verify foreign key references exist (profile IDs, etc.)

### **Cause 2: Foreign Key Constraint**

#### **Symptoms:**

- Cannot create subscriber
- Error: "key\_set\_id does not exist"
- Referenced entity not found

#### **Diagnostic Steps:**

1. Identify which foreign key is failing
2. Verify referenced entity exists:
  - key\_set\_id → key sets
  - epc\_profile\_id → EPC profiles
  - ims\_profile\_id → IMS profiles

#### **Solution:**

- Create referenced entity first
- Or use existing entity ID
- Follow [complete provisioning workflow](#)

### **Cause 3: Database Connectivity**

#### **Symptoms:**

- 500 errors
- All API calls fail
- Database connection errors

## Solution:

- See [Database Issues](#)
- 

# Diagnostic Tools and Commands

## Control Panel Quick Checks

### 1. System Overview

- URL: `https://[hostname]:7443/overview`
- Check: Subscriber counts, active sessions, system status

### 2. Diameter Status

- URL: `https://[hostname]:7443/diameter`
- Check: All critical peers connected

### 3. Application Health

- URL: `https://[hostname]:7443/application`
- Check: Memory usage, process count, uptime

## API Diagnostic Commands

### Check System Health:

```
curl -k https://hss.example.com:8443/api/status
```

### Query Subscriber:

```
# By IMSI
curl -k
https://hss.example.com:8443/api/subscriber/imsi/001001123456789

# By MSISDN
curl -k
https://hss.example.com:8443/api/subscriber/msisdn/14155551234

# By ID
curl -k https://hss.example.com:8443/api/subscriber/1
```

### List All Subscribers:

```
curl -k https://hss.example.com:8443/api/subscriber
```

### Check Profile Configuration:

```
# EPC Profile
curl -k https://hss.example.com:8443/api/epc/profile/1

# IMS Profile
curl -k https://hss.example.com:8443/api/ims/profile/1

# Roaming Profile
curl -k https://hss.example.com:8443/api/roaming/profile/1
```

## Network Diagnostic Commands

### Test Diameter Port Connectivity:

```
telnet [PEER_IP] 3868
```

### Check TLS Certificate:

```
openssl s_client -connect [hostname]:8443 -showcerts
```

## Test Database Connectivity:

```
psql -h [DB_HOST] -U [DB_USER] -d [DB_NAME] -c "SELECT COUNT(*)  
FROM subscriber;"
```

## Log Analysis

### Search Logs for Specific IMSI:

```
grep "001001123456789" /var/log/omnihss/omnihss.log
```

### Find Authentication Failures:

```
grep "authentication.*fail" /var/log/omnihss/omnihss.log
```

### Check Diameter Peer Events:

```
grep "Diameter peer" /var/log/omnihss/omnihss.log
```

### Find Database Errors:

```
grep -i "database.*error" /var/log/omnihss/omnihss.log
```

---

# Escalation Guidelines

## When to Escalate

Escalate to engineering/vendor support when:

1. **System-wide failures** that cannot be resolved with documented procedures

2. **Data corruption** or inconsistent database state
3. **Suspected software bugs** or unexpected behavior
4. **Performance issues** that cannot be resolved with tuning
5. **Security incidents** or unauthorized access
6. **Questions about undocumented behavior**

## Information to Provide

When escalating, include:

1. **Detailed symptoms** - What is failing, when, for whom
2. **Steps taken** - What troubleshooting you've already done
3. **Logs** - Relevant log excerpts showing the issue
4. **Configuration** - Relevant portions of runtime.exs (redact sensitive data)
5. **Environment** - OmniHSS version, database version, OS version
6. **Impact** - How many subscribers affected, business impact
7. **Subscriber examples** - Specific IMSIs showing the problem

## Critical vs Non-Critical

### Critical Issues (Escalate Immediately):

- System completely down
- All subscribers unable to attach
- Database corruption
- Security breach

### Non-Critical Issues (Document and Escalate During Business Hours):

- Single subscriber issues that can be worked around
  - Performance degradation that's manageable
  - Enhancement requests
  - Documentation questions
-

# Common Error Messages Reference

## Authentication Errors

Error Message	Cause	Solution
"Authentication vectors generation failed"	Missing or invalid key set	Check <a href="#">key set configuration</a>
"SQN synchronization failure"	SQN out of sync	<a href="#">Wait for resync</a>
"Subscriber not found"	Invalid IMSI	Verify IMSI, provision subscriber
"Subscriber disabled"	enabled=false	<a href="#">Enable subscriber</a>

## Diameter Errors

Error Message	Cause	Solution
"Diameter peer connection timeout"	Network issue	<a href="#">Check network connectivity</a>
"CER/CEA exchange failed"	Configuration mismatch	<a href="#">Verify Diameter config</a>
"Application not supported"	Peer doesn't support required app	<a href="#">Check peer applications</a>
"TLS handshake failed"	Certificate issue	<a href="#">Check certificates</a>

## Database Errors

Error Message	Cause	Solution
"Connection refused"	Database down	<a href="#">Start database</a>
"Authentication failed"	Wrong credentials	<a href="#">Fix credentials</a>
"No connections available"	Pool exhausted	<a href="#">Increase pool size</a>
"Query timeout"	Slow query	<a href="#">Optimize queries</a>

## API Errors

Error Message	Cause	Solution
"key_set_id does not exist"	Invalid foreign key	Create key set first
"IMSI has already been taken"	Duplicate IMSI	Use different IMSI or delete existing
"Validation error"	Invalid input	Check field format and requirements

---

[← Back to Operations Guide](#) | [Next: API Reference](#) →

# OmniHSS Webhook Integration

[← Back to Operations Guide](#)

---

## Table of Contents

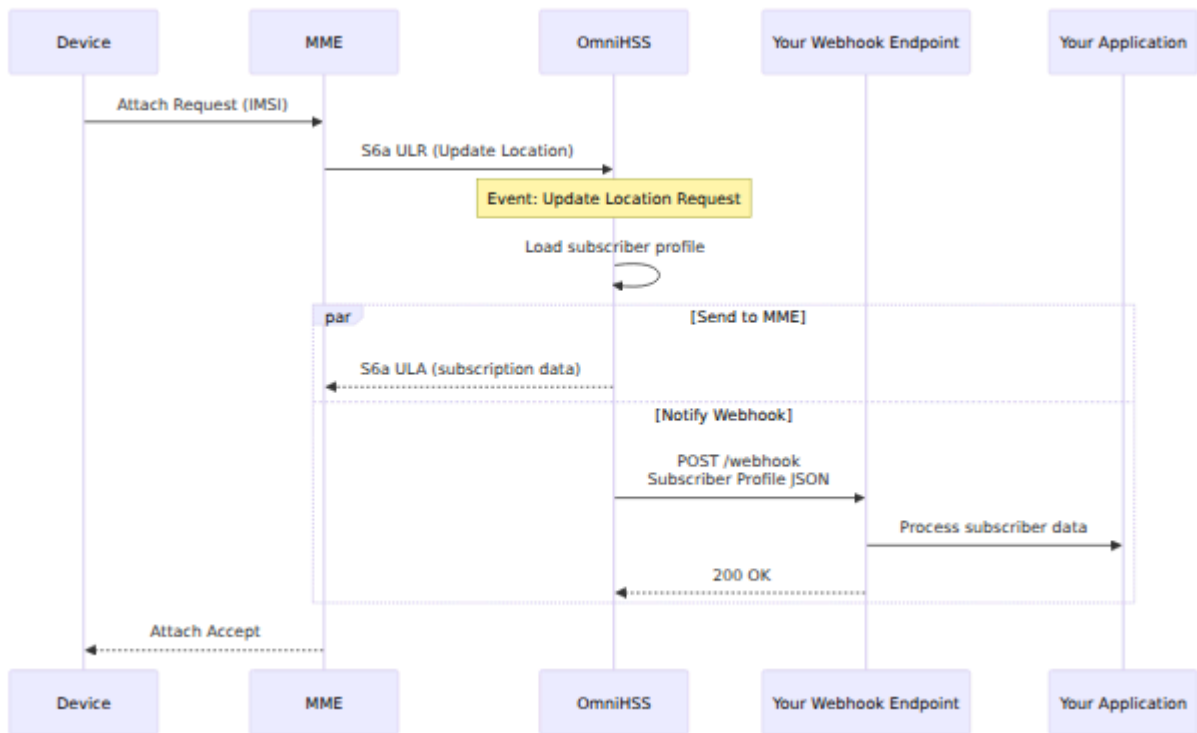
- [Overview](#)
  - [How Webhooks Work](#)
  - [Webhook Events](#)
  - [Webhook Payload](#)
  - [Configuration](#)
  - [Use Cases](#)
  - [Security Considerations](#)
  - [Troubleshooting](#)
- 

## Overview

OmniHSS supports **webhooks** to notify external systems about subscriber events in real-time. When specific events occur (such as location updates, authentication requests, or IMS registrations), OmniHSS can send an HTTP POST request to your configured webhook endpoint with the complete subscriber profile data.

## What Are Webhooks?

Webhooks are HTTP callbacks that allow OmniHSS to push event notifications to your application as they happen, rather than requiring your application to poll the HSS API for changes.



## Key Benefits

- **Real-time notifications** - Get instant updates when subscriber events occur
- **Complete subscriber data** - Each webhook includes the full subscriber profile (same as `GET /api/subscriber`)
- **Event-driven automation** - Trigger workflows, analytics, or provisioning based on network events
- **Reduced polling** - No need to continuously query the API for subscriber status changes
- **Integration flexibility** - Connect OmniHSS to billing systems, analytics platforms, or custom applications

## How Webhooks Work

### Event Flow

1. **Event occurs** - A subscriber performs an action (attach, location update, IMS registration, etc.)

2. **HSS processes event** - OmniHSS handles the Diameter request/response normally
3. **Webhook triggered** - If a webhook is registered for this event type, HSS sends HTTP POST to your endpoint
4. **Subscriber data included** - The webhook payload contains the complete subscriber profile as JSON
5. **Your application responds** - Your endpoint should return HTTP 200-299 to acknowledge receipt

## Delivery Guarantees

- **Best effort delivery** - Webhooks are sent asynchronously and do not block network operations
- **Timeout** - Webhook requests timeout after 5 seconds
- **No retries** - If your endpoint is unavailable or returns an error, the webhook is not retried
- **Order not guaranteed** - Events may arrive out of order under high load

**Important:** Network operations (authentication, location updates, etc.) are **not** dependent on webhook delivery. If your webhook endpoint is down, subscriber service continues normally.

---

## Webhook Events

OmniHSS can trigger webhooks for the following events:

## EPC/LTE Events

Event	Trigger	Description
<code>update_location_request</code>	S6a ULR	Subscriber attaches or performs Tracking Area Update
<code>authentication_information_request</code>	S6a AIR	Network requests authentication vectors for subscriber
<code>purge_request</code>	S6a PUR	MME removes subscriber context (device powered off, detached)
<code>cancel_location_answer</code>	S6a CLA	MME acknowledges subscriber deregistration

## IMS Events

Event	Trigger	Description
<code>ims_registration</code>	Cx SAR	Subscriber registers for IMS/VoLTE service
<code>ims_deregistration</code>	Cx SAR (de-reg)	Subscriber deregisters from IMS
<code>ims_profile_request</code>	Sh UDR	Application Server requests subscriber IMS profile

## Policy Events (PCRF)

Event	Trigger	Description
<code>policy_request</code>	Gx CCR	P-GW requests policy for subscriber data session
<code>media_authorization</code>	Rx AAR	P-CSCF requests media authorization for IMS call

## Multi-IMSI Events

Event	Trigger	Description
<code>imsi_switch</code>	ULR for different IMSI on same SIM	Device switches to different IMSI on multi-IMSI SIM

---

## Webhook Payload

### Request Format

When an event occurs, OmniHSS sends an HTTP POST request to your configured webhook URL:

```
POST /your-webhook-endpoint HTTP/1.1
Host: your-server.com
Content-Type: application/json
X-OmniHSS-Event: update_location_request
X-OmniHSS-Event-ID: 550e8400-e29b-41d4-a716-446655440000
X-OmniHSS-Timestamp: 2025-01-15T14:30:00Z
```

```
{
  "event": "update_location_request",
  "event_id": "550e8400-e29b-41d4-a716-446655440000",
  "timestamp": "2025-01-15T14:30:00Z",
  "subscriber": {
    "id": 1234,
    "imsi": "001001123456789",
    "enabled": true,
    "ims_enabled": true,
    "msisdns": [
      {"id": 1, "msisdn": "14155551001"},
      {"id": 2, "msisdn": "14155551002"}
    ],
    "sim": {
      "id": 5678,
      "iccid": "8991101200003204510",
      "is_esim": false
    },
    "key_set": {
      "id": 100,
      "amf": "8000"
    },
    "epc_profile": {
      "id": 1,
      "name": "Premium 100Mbps",
      "ue_ambr_dl_kbps": 100000,
      "ue_ambr_ul_kbps": 50000
    },
    "ims_profile": {
      "id": 1,
      "name": "Standard VoLTE"
    },
    "roaming_profile": {
      "id": 1,
      "name": "International Roaming Allowed"
    },
  },
}
```

```

"subscriber_state": {
  "mme_host": "mme-01.example.com",
  "mme_realm": "epc.mnc001.mcc001.3gppnetwork.org",
  "visited_plmn": "001001",
  "last_update": "2025-01-15T14:30:00Z"
},
"custom_attributes": {
  "account_type": "premium",
  "billing_plan": "unlimited"
}
},
"event_context": {
  "visited_plmn": "310410",
  "mme_host": "mme-roaming.example.com",
  "location_update_type": "initial_attach"
}
}

```

## Payload Structure

Field	Type	Description
<code>event</code>	string	Event type (e.g., <code>update_location_request</code> )
<code>event_id</code>	string	Unique UUID for this webhook delivery
<code>timestamp</code>	string	ISO 8601 timestamp when event occurred
<code>subscriber</code>	object	<b>Complete subscriber profile</b> (same as <code>GET /api/subscriber/:id</code> )
<code>event_context</code>	object	Additional event-specific context data

## Event Context Fields

The `event_context` object contains event-specific information:

**For `update_location_request`:**

```
{
  "visited_plmn": "310410",
  "mme_host": "mme-roaming.example.com",
  "mme_realm": "epc.mnc410.mcc310.3gppnetwork.org",
  "location_update_type": "initial_attach"
}
```

**For `imsi_switch`:**

```
{
  "previous_imsi": "001001111111111",
  "new_imsi": "310410222222222",
  "sim_id": 5678,
  "previous_mme_host": "mme-home.example.com",
  "new_mme_host": "mme-roaming.example.com"
}
```

**For `ims_registration`:**

```
{
  "scscf_host": "scscf-01.ims.example.com",
  "public_identities": [
    "sip:001001123456789@ims.mnc001.mcc001.3gppnetwork.org",
    "sip:+14155551001@ims.example.com",
    "tel:+14155551001"
  ]
}
```

# HTTP Headers

Header	Description	Example
Content-Type	Always application/json	application/json
X-OmniHSS-Event	Event type	update_location_request
X-OmniHSS-Event-ID	Unique event identifier	UUID
X-OmniHSS-Timestamp	Event timestamp	ISO 8601 format
User-Agent	OmniHSS version	OmniHSS/1.0

---

## Configuration

### Registering Webhooks

Webhooks are configured via the OmniHSS API.

#### Register a Webhook

```
curl -k -X POST https://hss.example.com:8443/api/webhook \
-H "Content-Type: application/json" \
-d '{
  "webhook": {
    "url": "https://your-server.com/omnihss-webhook",
    "events": [
      "update_location_request",
      "ims_registration",
      "imsi_switch"
    ],
    "enabled": true,
    "description": "Production billing system webhook"
  }
}'
```

### Response:

```
{
  "data": {
    "id": 1,
    "url": "https://your-server.com/omnihss-webhook",
    "events": [
      "update_location_request",
      "ims_registration",
      "imsi_switch"
    ],
    "enabled": true,
    "description": "Production billing system webhook",
    "created_at": "2025-01-15T14:00:00Z"
  }
}
```

### List Webhooks

```
curl -k https://hss.example.com:8443/api/webhook
```

### Update Webhook

```
curl -k -X PUT https://hss.example.com:8443/api/webhook/1 \  
-H "Content-Type: application/json" \  
-d '{  
  "webhook": {  
    "enabled": false  
  }  
'
```

## Delete Webhook

```
curl -k -X DELETE https://hss.example.com:8443/api/webhook/1
```

## Webhook Endpoint Requirements

Your webhook endpoint must:

1. **Accept POST requests** with `Content-Type: application/json`
2. **Respond quickly** - Return HTTP 200-299 within 5 seconds
3. **Be idempotent** - Handle duplicate deliveries gracefully
4. **Use HTTPS** - For security, use TLS/SSL endpoints (recommended)
5. **Validate payloads** - Verify the request is from OmniHSS (see Security section)

### Example Webhook Handler (Node.js/Express):

```
const express = require('express');
const app = express();

app.post('/omnihss-webhook', express.json(), (req, res) => {
  const { event, subscriber, event_context } = req.body;

  console.log(`Received event: ${event}`);
  console.log(`Subscriber IMSI: ${subscriber.imsi}`);

  // Process the subscriber data
  // ... your business logic here ...

  // Respond immediately to acknowledge receipt
  res.status(200).json({ received: true });

  // Handle async processing after response
  processWebhook(req.body).catch(console.error);
});

async function processWebhook(payload) {
  // Your async processing logic
  // e.g., update billing system, trigger analytics, etc.
}

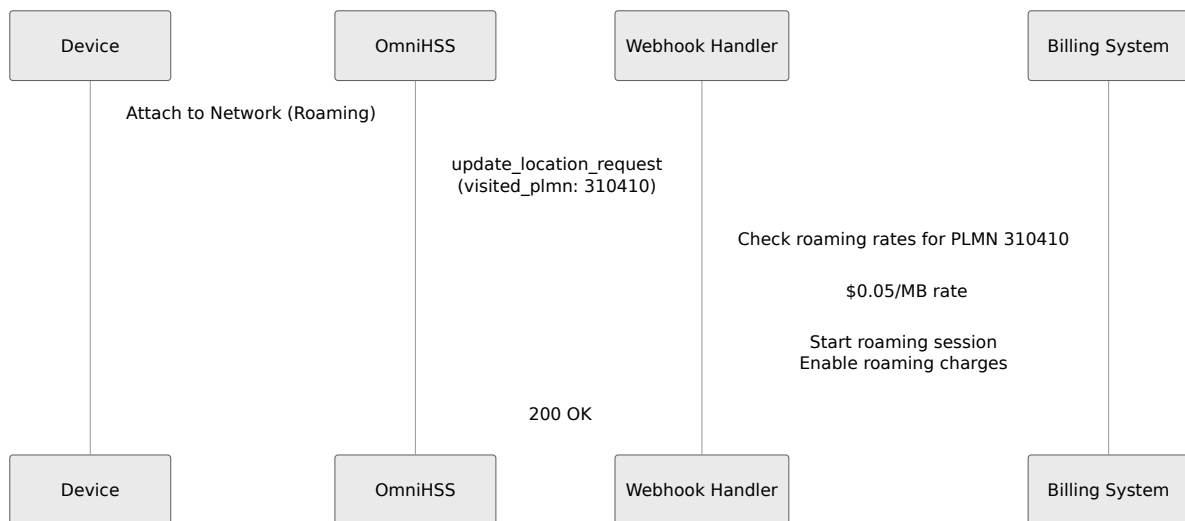
app.listen(3000);
```

---

# Use Cases

## 1. Real-Time Billing and Usage Tracking

Track subscriber network usage and trigger billing events in real-time.



### Benefits:

- Instantly detect when subscribers roam internationally
- Apply appropriate roaming charges in real-time
- Track session start/end times accurately
- Generate usage alerts when thresholds are reached

## 2. Analytics and Monitoring

Feed subscriber activity data into analytics platforms for real-time dashboards and reporting.

**Use Case:** Track active subscribers by region

```
// Webhook handler feeding data to analytics platform
app.post('/omnihss-webhook', async (req, res) => {
  const { event, subscriber, event_context } = req.body;

  if (event === 'update_location_request') {
    await analytics.track({
      event: 'subscriber_location_update',
      imsi: subscriber.imsi,
      visited_plmn: event_context.visited_plmn,
      timestamp: req.body.timestamp,
      profile: subscriber.epc_profile.name
    });
  }

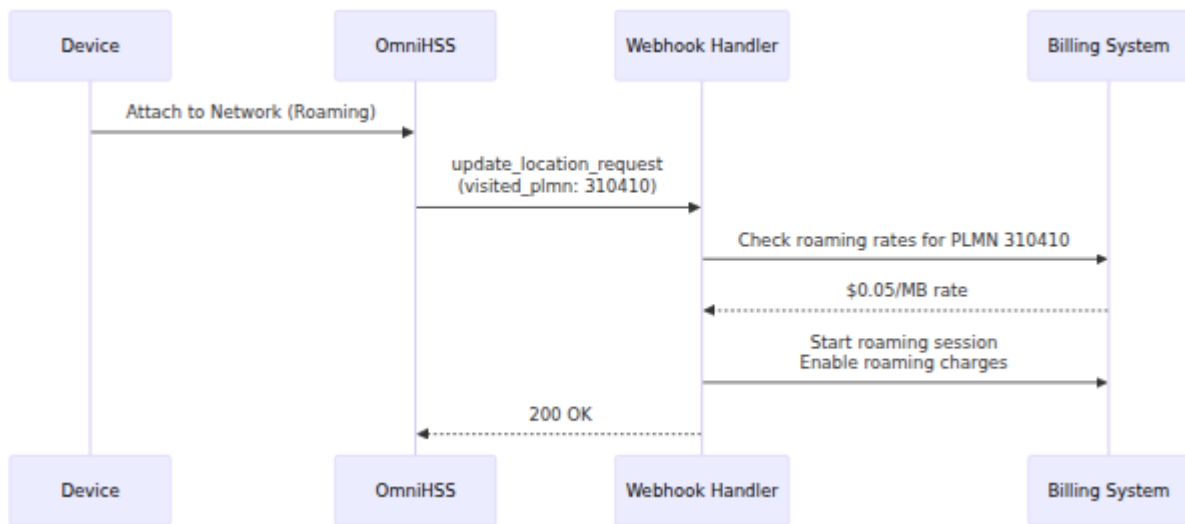
  res.status(200).send();
});
```

### **Analytics Dashboard:**

- Active subscribers per MME
- Roaming subscribers by country
- Service tier distribution
- IMS registration success rates

## **3. Fraud Detection and Security**

Detect suspicious activity patterns in real-time and trigger automated responses.



## Fraud Detection Scenarios:

### 1. Rapid Location Changes

- Subscriber attaches in Country A
- 30 minutes later, attaches in Country B (physically impossible)
- Action: Flag account, send alert to security team

### 2. IMSI Switch Abuse

- Multiple rapid IMSI switches on same SIM
- Possible SIM cloning or unauthorized multi-IMSI usage
- Action: Disable all IMSIs on SIM, notify fraud team

### 3. Unauthorized Roaming

- Subscriber roams to blocked country (sanctions, fraud risk)
- Action: Auto-disable subscriber, block network access

## Example Implementation:

```

@app.route('/omnihss-webhook', methods=['POST'])
def webhook_handler():
    data = request.json
    subscriber = data['subscriber']
    event_context = data.get('event_context', {})

    if data['event'] == 'update_location_request':
        visited_plmn = event_context.get('visited_plmn')

        # Check for blocked countries
        if visited_plmn in BLOCKED_PLMNS:
            disable_subscriber(subscriber['imsi'])
            alert_security_team(subscriber, 'Roaming to blocked
PLMN')

        # Check for impossible travel
        if is_impossible_travel(subscriber['imsi'], visited_plmn):
            flag_for_review(subscriber['imsi'])
            alert_fraud_team(subscriber, 'Impossible travel
detected')

    return jsonify({'status': 'ok'}), 200

```

## 4. Provisioning Automation

Automatically provision or update subscriber services based on network events.

**Use Case:** Auto-enable IMS when subscriber first uses VoLTE

```

app.post('/omnihss-webhook', async (req, res) => {
  const { event, subscriber } = req.body;

  if (event === 'ims_registration' && !subscriber.ims_enabled) {
    // First-time IMS user - enable IMS permanently
    await omnihss.updateSubscriber(subscriber.id, {
      ims_enabled: true,
      custom_attributes: {
        ...subscriber.custom_attributes,
        volte_activated_at: new Date().toISOString()
      }
    });

    // Update CRM
    await crm.updateCustomer(subscriber.imsi, {
      features: ['volte']
    });
  }

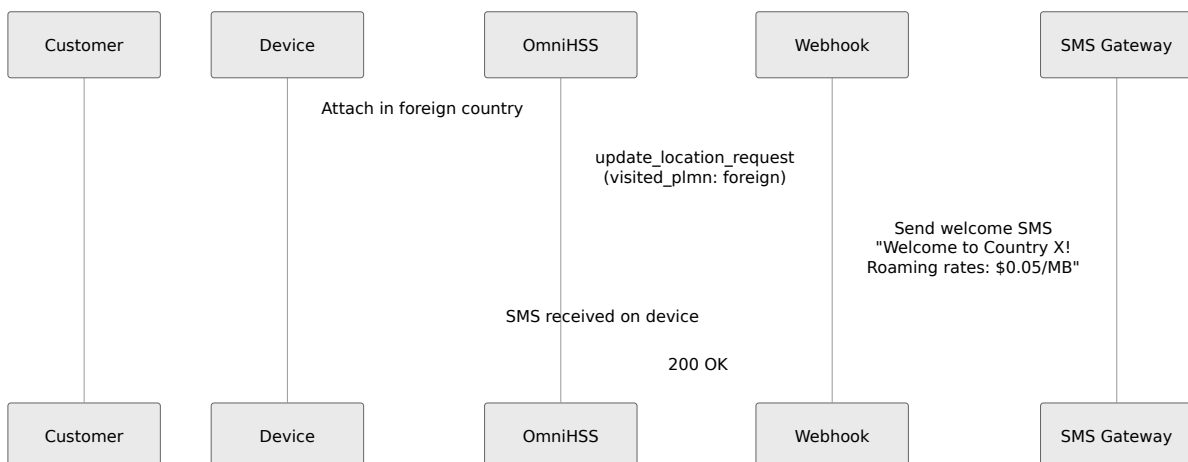
  res.status(200).send();
});

```

## 5. Customer Notifications

Send real-time notifications to customers about their service.

**Use Case:** Welcome message when roaming internationally



**Example Notifications:**

- "Welcome to [Country]! Roaming rates apply."
- "You've used 80% of your data allowance"
- "VoLTE service now active on your device"
- "Your account has been upgraded to Premium"

## 6. Multi-IMSI SIM Management

Track and manage subscribers with multi-IMSI SIMs, receiving notifications when they switch IMSIs.

```
app.post('/omnihss-webhook', async (req, res) => {
  const { event, subscriber, event_context } = req.body;

  if (event === 'imsi_switch') {
    const { previous_imsi, new_imsi, sim_id } = event_context;

    // Log IMSI switch for analytics
    await db.logImsiSwitch({
      sim_id,
      from_imsi: previous_imsi,
      to_imsi: new_imsi,
      timestamp: req.body.timestamp
    });

    // Update billing system
    await billing.endSession(previous_imsi);
    await billing.startSession(new_imsi);

    // Alert if excessive switching (potential fraud)
    const switchCount = await db.getSwitchCount(sim_id, '24h');
    if (switchCount > 10) {
      await alertFraudTeam(`Excessive IMSI switching: SIM
${sim_id}`);
    }
  }

  res.status(200).send();
});
```

## 7. Integration with External Systems

Connect OmniHSS to third-party systems without polling.

### Example Integrations:

- **CRM Systems** - Update customer records with service usage
  - **Network Monitoring** - Feed subscriber data to network analytics platforms
  - **Billing Systems** - Trigger charges based on network events
  - **Ticketing Systems** - Auto-create tickets for failed authentications
  - **Data Warehouses** - Stream subscriber events for big data analysis
- 

## Security Considerations

### Webhook Secret/Signature

To verify webhooks are from OmniHSS, implement signature verification:

```
# Configure webhook with secret
curl -k -X POST https://hss.example.com:8443/api/webhook \
  -H "Content-Type: application/json" \
  -d '{
    "webhook": {
      "url": "https://your-server.com/omnihss-webhook",
      "events": ["update_location_request"],
      "secret": "your-secret-key-here"
    }
  }'
```

OmniHSS will include an `X-OmniHSS-Signature` header:

```
X-OmniHSS-Signature:
sha256=5d7a8f9b2c1e3a4d6f7e8b9c0a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a
```

## Verify the signature:

```
const crypto = require('crypto');

function verifyWebhook(req) {
  const signature = req.headers['x-omnihss-signature'];
  const secret = process.env.WEBHOOK_SECRET;
  const payload = JSON.stringify(req.body);

  const expectedSignature = 'sha256=' +
    crypto.createHmac('sha256', secret)
      .update(payload)
      .digest('hex');

  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expectedSignature)
  );
}

app.post('/omnihss-webhook', (req, res) => {
  if (!verifyWebhook(req)) {
    return res.status(401).json({ error: 'Invalid signature' });
  }

  // Process webhook...
  res.status(200).send();
});
```

## Best Practices

1. **Use HTTPS** - Always use TLS for webhook endpoints
2. **Validate signatures** - Verify webhook signatures to prevent spoofing
3. **Rate limiting** - Implement rate limiting on webhook endpoints
4. **IP allowlisting** - Restrict webhook access to OmniHSS IP addresses
5. **Monitor failures** - Track webhook delivery failures and errors
6. **Sanitize data** - Validate and sanitize webhook payloads before processing
7. **Secure credentials** - Store webhook secrets in secure configuration (environment variables, secrets manager)

# Data Privacy

Webhook payloads contain **sensitive subscriber information**:

- IMSI (subscriber identity)
- MSISDNs (phone numbers)
- Location data (visited PLMN, MME)
- Service profile information

## Compliance Requirements:

- **GDPR** - Ensure webhook data is processed in compliance with GDPR
  - **Data retention** - Implement appropriate data retention policies
  - **Access control** - Restrict webhook endpoint access
  - **Encryption** - Use TLS for webhook transport
  - **Audit logging** - Log all webhook deliveries for compliance
- 

# Troubleshooting

## Webhook Not Received

### Symptoms:

- Events occur but webhook is not triggered
- Webhook endpoint never receives requests

### Troubleshooting Steps:

#### 1. Verify webhook is enabled:

```
curl -k https://hss.example.com:8443/api/webhook  
# Check "enabled": true
```

#### 2. Check webhook events configuration:

- Ensure the event type is included in the webhook's `events` list
- Example: If you want `ims_registration` events, verify it's in the events array

### 3. Review HSS logs:

- Check for webhook delivery errors
- Look for network connectivity issues
- Verify no DNS resolution failures

### 4. Test endpoint accessibility:

```
curl -X POST https://your-server.com/omnihss-webhook \  
-H "Content-Type: application/json" \  
-d '{"test": true}'
```

## Webhook Timing Out

### Symptoms:

- HSS logs show webhook timeout errors
- Webhook endpoint receives request but HSS marks as failed

### Solution:

#### 1. Respond immediately:

- Return HTTP 200 within 5 seconds
- Process data asynchronously after responding

#### 2. Optimize endpoint performance:

```
// BAD - Slow synchronous processing
app.post('/webhook', (req, res) => {
  processData(req.body); // Blocks for 10 seconds
  res.status(200).send();
});

// GOOD - Async processing after response
app.post('/webhook', (req, res) => {
  res.status(200).send(); // Respond immediately
  processData(req.body); // Process async
});
```

## Duplicate Webhooks

### Symptoms:

- Same event delivered multiple times
- `event_id` is identical for duplicate deliveries

### Cause:

- Network retries (though OmniHSS doesn't retry, network infrastructure might)
- Multiple webhooks registered for same event

### Solution:

Implement idempotency using `event_id`:

```
const processedEvents = new Set();

app.post('/omnihss-webhook', (req, res) => {
  const eventId = req.body.event_id;

  if (processedEvents.has(eventId)) {
    // Already processed, skip
    return res.status(200).json({ status: 'duplicate' });
  }

  processedEvents.add(eventId);

  // Process webhook...
  processWebhook(req.body);

  res.status(200).json({ status: 'processed' });
});
```

## Webhook Returns Error

### Symptoms:

- Endpoint returns HTTP 4xx or 5xx
- HSS logs webhook delivery failure

### Common Errors:

1. **401 Unauthorized** - Signature verification failed
  - Check webhook secret matches configuration
  - Verify signature calculation algorithm
2. **400 Bad Request** - Invalid payload
  - Check webhook payload parsing
  - Ensure Content-Type header is handled
3. **500 Internal Server Error** - Endpoint crashed
  - Review endpoint error logs

- Add error handling and logging

## Solution:

Add comprehensive error handling:

```
app.post('/omnihss-webhook', async (req, res) => {
  try {
    // Verify signature
    if (!verifyWebhook(req)) {
      return res.status(401).json({ error: 'Invalid signature' });
    }

    // Validate payload
    if (!req.body.event || !req.body.subscriber) {
      return res.status(400).json({ error: 'Invalid payload' });
    }

    // Process webhook
    await processWebhook(req.body);

    res.status(200).json({ status: 'ok' });

  } catch (error) {
    console.error('Webhook processing error:', error);
    // Return 200 to prevent retry, log error for investigation
    res.status(200).json({ status: 'error', message: error.message
  });
  }
});
```

## Missing Subscriber Data

### Symptoms:

- Webhook received but subscriber object is incomplete
- Expected fields are null or missing

### Possible Causes:

1. **Subscriber not fully provisioned** - Some profiles may be optional (IMS, roaming)
2. **Data race condition** - Subscriber updated between event trigger and webhook send

### **Solution:**

Handle optional fields gracefully:

```
const { subscriber } = req.body;

// Check for optional fields
const imsProfile = subscriber.ims_profile || { name: 'No IMS' };
const roamingProfile = subscriber.roaming_profile || { name: 'No Roaming' };

// Handle missing MSISDNs
const msisdns = subscriber.msisdns || [];
```

---

# Monitoring and Observability

## Webhook Metrics

Track webhook performance and reliability:

### **Metrics to Monitor:**

- Webhook delivery rate (successful vs. failed)
- Webhook latency (time from event to endpoint response)
- Endpoint response times
- Error rates by endpoint
- Events per second

### **Example Dashboard Query (Prometheus/Grafana):**

```
# Webhook success rate
rate(omnihss_webhook_success_total[5m]) /
rate(omnihss_webhook_attempts_total[5m])

# Webhook latency
histogram_quantile(0.95, omnihss_webhook_duration_seconds)
```

## Webhook Logs

Enable detailed webhook logging for troubleshooting:

### Log Format:

```
{
  "timestamp": "2025-01-15T14:30:00Z",
  "level": "info",
  "component": "webhook",
  "event_id": "550e8400-e29b-41d4-a716-446655440000",
  "webhook_id": 1,
  "event_type": "update_location_request",
  "subscriber_imsi": "001001123456789",
  "endpoint": "https://your-server.com/omnihss-webhook",
  "http_status": 200,
  "duration_ms": 145,
  "error": null
}
```

---

[← Back to Operations Guide](#) | [Next: API Reference](#) →