ASAC.01-Seamless Authenticator subsystem enhancement for TS.43 Operator Token

Version 1.2

31 March 2025

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Copyright © 2025 GSM Association

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Compliance Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

This Permanent Reference Document is classified by GSMA as an Industry Specification, as such it has been developed and is maintained by GSMA in accordance with the provisions set out in GSMA AA.35 – Procedures for Industry Specifications.

Table of Contents

1 Introduction 4
1.1 Overview 4
1.2 Scope 4
1.3 Definitions 4
1.4 Abbreviations. 5
1.5 References 5
2 TS.43 Operator Token 6
3 High level architecture 8
4 Generic service flow 11
4.1 Eligibility Token Generation 11
4.2 Eligibility Token consumption & Operator Token generation 15
4.3 Operator Token consumption 21
4.4 Eligibility Token Payload (JWT) 22
4.5 Operator Token 24
4.6 Operator Token scopes 24
4.7 Operator Token Mapping 25
4.8 Aggregator Response – step 5 25
4.9 3rd Party App Server Response – step 7 26
4.10 TS.43 Client 302 Redirect Response parameters – step 21 26
4.11 TS.43 Client 200 Response parameters – step 23 26
5 Interfaces characteristics 26
5.1 Authorization Server and Entitlement Configuration Server interface 26
5.1.1 Authorization Server High level requirements 27
5.2 3rd party application and TS.43 Client interface 27
5.2.1 TS.43 Client High level requirements 27
5.3 Authorization Server and Aggregator interface 27
5.3.1 Authorization Server High level requirements 28
6 Use cases Characteristics 28
6.1 Verifying a phone number provided by the user within an account registration flow (Mobile Connect Number Verify) 28
6.1.1 Eligibility Token Generation 29
6.1.2 Eligibility Token consumption & Operator Token generation 32
6.1.3 Operator Token consumption & Number Verify flow 36
6.2 MNO Native Application use case 39
6.2.1 High level architecture without aggregator 40
6.2.2 MNO Native Application use case flow 43
6.2.3 TS.43 client URI-scheme – step [003] 55
6.2.4 OperatorToken (JWT) – step [013] 56
6.2.5 Entitlement Configuration Server High level requirements 57
6.3 Open Gateway use case 57
6.3.1 Overview 57
6.3.2 Interface flow (sequence diagram) – AccessToken consumption 59

6.3.3 Functional requirements 60
Annex A Document Management 61
A.1 Document History 61
A.2 Other Information 61

1 Introduction

1.1 Overview

Customers value seamless & secure authentication solutions, like MSISDN Seamless Authentication which enables a secure and silent authentication process using MNO assets.

As this MSISDN Seamless Authentication relies on network enablers, the main drawbacks are:

- Only works when the mobile device is connected to the Internet via Mobile Network not via WiFi
- Is only guaranteed to work from native applications, not from web clients
- It is hard to certify at a high level of security

The challenge therefore is to implement a secure (certifiable) MSISDN seamless authentication including when initiated from device web clients whereas the user's mobile device is using Wi-Fi connectivity to access Internet Services.

It is required to extend TS.43 to enable MSISDN seamless authentication via Wi-Fi and trigger MSISDN seamless authentication.

1.2 Scope

In Scope	Out of Scope
• Authorization Server and Entitlements Configuration Server interface • Authorization Server and 3rd Party application interface • 3rd party application and TS.43 client interface	• Authorization Server and TS.43 client interface • TS.43 client capability to authenticate 3rd party application

1.3 Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [5] [6] when, and only when, they appear in all capitals, as shown here.

Term	Description
Aggregator or Channel Partner (CP)	means a third party to which MNO delivers the OperatorToken capability which then further enables this capability to Service Providers e.g.: hyperscalers, Open Gateway provider.
Authorization Server (AS)	The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

Eligibility Token	As defined in RFC 7519 [06] JWT, that is produced by Channel Partner (CP), upon successful SP client authentication. Eligibility Token is consumed by MNOs AS (either directly from TS.43 client or via ECS) and triggers OperatorToken creation.
Service Provider (SP)	means an individual or entity that utilizes the Services on a commercial basis in order to deliver its own service to Subscribers.

Table 1 Definitions

1.4 Abbreviations.

Term	Description
API	Application Programming Interface
AS	Authorisation Server
CP	Channel Partner
eIDAS	electronic Identification, Authentication and trust Services
ECS	Entitlement Configuration Server
HTTP	Hyper Text Transport Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token, according to RFC 7519 [6]
MC	MobileConnect
MNO	Mobile Network Operator
MSISDN	Mobile Station International Subscriber Directory Number
OIDC	Open,ID Connect
OGW	Open Gateway
RP	Relying Party: The application/service that needs the authentication and API services
SP	Service Provider or 3rd party application

Table 2 Abbreviations

1.5 References

Ref	Doc Number	Title
[1]	TS.43	Service Entitlement Configuration v11 https://www.gsma.com/newsroom/wp-content/uploads//TS.43-v11.0-Service-Entitlement-Configuration.pdf
[2]	OpenID Connect Core Specification	“An interoperable authentication protocol based on the Oauth 2.0 family of specifications” available at

Ref	Doc Number	Title
		https://openid.net/specs/openid-connect-core-1_0.html
[3]	OIDF CIBA	OpenID Connect MODRNA Client Initiated Backchannel Authentication Flow 1.0 https://openid.net/specs/openid-connect-modrna-client-initiated-backchannel-authentication-1_0.html
[4]	RFC 6749	"The Oauth 2.0 Authorization Framework", D. Hardt, Ed. October 2012 available at http://www.ietf.org/rfc/rfc6749.txt
[5]	RFC 6750	M. Jones and D. Hardt, "The Oauth 2.0 Authorization Framework: Bearer Token Usage," RFC 6750, October 2012 https://tools.ietf.org/html/rfc6750
[6]	RFC 7519	JSON Web Token (JWT), https://datatracker.ietf.org/doc/html/rfc7519
[7]	RFC 9068	JSON Web Token (JWT) Profile for Oauth 2.0 Access Tokens, https://datatracker.ietf.org/doc/rfc9068/
[8]	IDY.01	Mobile Connect device initiated OIDC profile Version 3.0 https://www.gsma.com/identity/wp-content/uploads/2022/11/IDY.01-v3.0.pdf
[9]	IDY.04	Mobile Connect Core Technical Requirements Version 1.2 https://www.gsma.com/identity/wp-content/uploads/2022/11/IDY.04-v1.2.pdf
[10]	IDY.54	Mobile Connect Verified MSISDN Definition and Technical Requirements Version 1.0 https://www.gsma.com/identity/wp-content/uploads/2022/12/IDY.54-Mobile-Connect-Verified-MSISDN-Definition-and-Technical-Requirements-1.0.pdf
[11]	PKCE	Proof Key for Code Exchange by OAuth Public Clients https://www.rfc-editor.org/rfc/rfc7636

Table 3 References

2 TS.43 Operator Token

Customers value seamless & secure authentication solutions, like MSISDN Seamless Authentication which enables a secure and silent authentication process using MNO assets. The existing drawbacks of MSISDN Seamless Authentication solutions available on the market that rely on network enablers are:

- Only works when the mobile device is connected to the Internet via Mobile Network not via WiFi
- Is only guaranteed to work from native applications, not from device web clients
- It is hard to certify at a high level of security

Operator Token acquisition process, as defined in TS.43[1], implements a secure (certifiable) authentication solution, when the user's mobile device consumes 3rd party applications.

The following presents the case where:

- Mobile operating system grants access to some methods to applications, one example being EAP-AKA authentication.
- A 3rd party application is installed on the device and does not have access to API allowing the use of EAP-AKA authentication.
- The TS.43 client has access to those methods
- The 3rd party and the ECS previously exchanged security information enabling authentication of the 3rd party application to the ECS. This could be done directly between ECS and application-backend or with a dedicated partner management system.
- The 3rd party application can obtain an access_token for the AcquireOperatorToken operation.

Note: This document calls the TS.43 access_token, EligibilityToken.

TS.43 mentions that ECS may implement additional features to enable enhanced access control mechanisms over partner/3rd party application management system (e.g. Authorization Server).

This specification aims to define suitable API and a native OS Client that would enable 3rd party applications to have a universal and SIM based authentication method that follows these principles:

- The service works on any IP bearer providing connectivity to the public internet (e.g. Mobile Data, WiFi).
- An OS API is available to applications and web browsers to allow triggering the MSISDN Seamless Authentication.
- Mobile Network Operator is maintaining the online access control list to the service
- The service will authenticate the MSISDN through an EAP-AKA transaction, leveraging a native OS EAP-AKA client, without the need for any additional application.
- The solution fits in a role of MNO authenticator as a part of a wider OpenID Connect flow towards the MNO.
- In the case of dual SIM, the user is able to choose which SIM card (or eSIM profile) is used by default. Optionally, the user will be able to decide which SIM to use for each application.
- EAP-AKA triggering is real-time and fast in a way that it does not affect the user experience.
- EAP-AKA triggering is secure (no man-in-the-middle attacks, no DDOS attacks)
- Only the needed functionality for the trigger will be exposed. 3rd parties do not get full access to the SIM.
- Personal Identifiable Information, like the MSISDN, is always shared in a way that ensures unlinkability and confidentiality.
- The TS.43 client knows which ECS is responsible for the (selected) SIM
- The solution complies with any applicable data protection laws (like GDPR) and regulation of telecommunications services (including telecom secrecy and lawful interception).

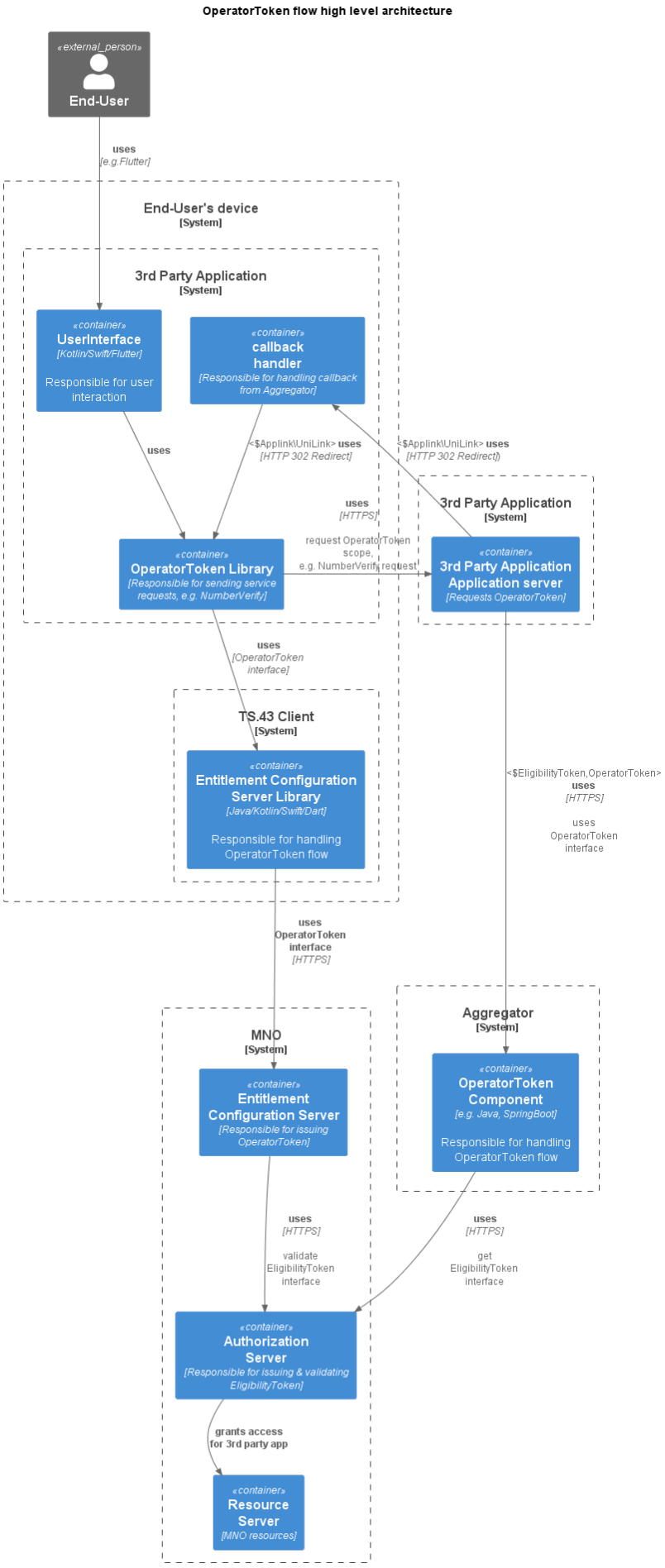
TS.43 Operator Token consumption can support following use cases:

- Verifying a phone number provided by the user within an account registration flow (such as: Mobile Connect Number Verify and CAMARA Number Verification API).
- Anti-fraud check (Camara SimSwap API),
- Second factor authentication without requiring user interaction (Mobile Connect Authenticate).
- Retrieve the necessary authentication material to achieve a high level of security certification for Identity Wallets (eIDAS 2.0).

3 High level architecture

```
@startuml OperatorToken flow high level architecture

!include  https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Container.puml

title OperatorToken flow high level architecture

Person_Ext(user, "End-User")

System_Boundary(app, "3rd Party Application") {
    Container(appServer, "3rd Party Application\nApplication server", "Requests
OperatorToken")
}

System_Boundary(mno, "MNO") {
    Container(es, "Entitlement\nConfiguration Server", "Responsible for issuing
OperatorToken")
    Container(authServ, "Authorization\nServer", "Responsible for issuing &
validating EligibilityToken")
    Container(resource, "Resource\nServer", "MNO resources")
}


System_Boundary(smartPhone, "End-User's device") {
    System_Boundary(ts43app, "TS.43 Client") {
        Container(esLibrary, "Entitlement Configuration Server Library",
"Java/Kotlin/Swift/Dart", "Responsible for handling OperatorToken flow")
    }
    System_Boundary(appFront, "3rd Party Application") {
        Container(ui, "UserInterface", "Kotlin/Swift/Flutter", "Responsible for
user interaction")
        Container(otLib, "OperatorToken Library", "Responsible for sending service
requests, e.g. NumberVerify")
        Container(callback, "callback\nhandler", "Responsible for handling callback
from Aggregator")
    }
}

System_Boundary(aggregator, "Aggregator") {
    Container(core, "OperatorToken Component", "e.g. Java, SpringBoot",
"Responsible for handling OperatorToken flow")
}

Rel(user, ui, "uses", "e.g.Flutter")
Rel(ui, otLib, "uses")
Rel(otLib, appServer, "uses", "HTTPS", "request OperatorToken scope, e.g.
NumberVerify request")
Rel(appServer, core, "uses", "HTTPS",
"uses\nOperatorToken\ninterface","EligibilityToken,OperatorToken")
Rel(callback, otLib, "uses","HTTP 302 Redirect","","Applink\UniLink")
```

```
Rel(otLib, esLibrary, "uses","OperatorToken\ninterface","","AppLink/UniLink")
Rel(core,authServ,"uses","HTTPS","get\nEligibilityToken\ninterface","EligibilityTok
en")
Rel(esLibrary,es, "uses\nOperatorToken\ninterface","HTTPS")
Rel(es, authServ, "uses","HTTPS","validate\nEligibilityToken\ninterface")
Rel(authServ,resource, "grants access\nfor 3rd party app")
Rel(appServer,callback,"uses","HTTP 302 Redirect","","Applink\UniLink"))

Lay_R(es, authServ)
Lay_D(authServ, resource)
Lay_R(authServ, aggregator)
Lay_D(otLib, authServ)
Lay_D(es, authServ)
Lay_R(otLib, appServer)
Lay_D(appServer,esLibrary)
@enduml
```


OperatorToken flow high level architecture
«external_person»
End-User
uses
[e.g.Flutter]
End-User's device
[System]
3rd Party Application
[System]
«container»
UserInterface
[Kotlin/Swift/Flutter]
Responsible for user interaction
«container»
callback handler
[Responsible for handling callback from Aggregator]
uses
<$Applink\UniLink> uses
[HTTP 302 Redirect]
<$Applink\UniLink> uses
[HTTP 302 Redirect])
uses
[HTTPS]
3rd Party Application
[System]
«container»
OperatorToken Library
[Responsible for sending service requests, e.g. NumberVerify]
request OperatorToken
scope,
e.g. NumberVerify request
«container»
3rd Party Application Application server
[Requests OperatorToken]
uses
[OperatorToken interface]
TS.43 Client
[System]
«container»
Entitlement Configuration Server Library
[Java/Kotlin/Swift/Dart]
Responsible for handling OperatorToken flow
<$EligibilityToken,OperatorToken>
uses
[HTTPS]
uses
OperatorToken interface
uses
OperatorToken interface
[HTTPS]
MNO
[System]
Aggregator
[System]
«container»
Entitlement Configuration Server
[Responsible for issuing OperatorToken]
«container»
OperatorToken Component
[e.g. Java, SpringBoot]
Responsible for handling OperatorToken flow
uses
[HTTPS]
validate EligibilityToken interface
uses
[HTTPS]
get EligibilityToken interface
«container»
Authorization Server
[Responsible for issuing & validating EligibilityToken]
grants access for 3rd party app
«container»
Resource Server
[MNO resources]

4 Generic service flow

This section describes generic flow for seamless authenticator subsystem enhancement for TS.43 Operator Token [1] that consists of

- Eligibility Token Generation
- Eligibility Token Consumption & Operator Token Generation
- Operator Token Consumption

4.1 Eligibility Token Generation

```
@startuml OperatorToken Flow with TS.43 Client – Generic – Eligibility Token
Generation

title OperatorToken flow with TS.43 Client for 3rd party apps (Eligibility Token
Generation)

!pragma teoz true

autonumber "<b>[000]"

actor "End-User" as user

box "User's device"
    participant "3rd party App" as app
    participant "TS.43 Client" as client <<carrier privileges>>
end box

box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box

participant "3rd party App\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

app -> server: requestOperatorTokenScope()
server -> aggregator: requestEligibilityForOperatorTokenScope(SpDetails)

rnote over aggregator
Aggregator prerequisites:
* expose jwks endpoint
* has to to provision SP to MNOs
* SP provisioning process TBD
(reference: numbering ranges announcement process)
end note

aggregator -> aggregator: validateRequest(SpDetails)

rnote over aggregator
* aggregator calls MNO to get JWT (option)
* this could be a random MNO to build trust (MNO trust circle)
end rnote

aggregator -> aggregator: craftEligibilityToken()

rnote over aggregator
signed with Aggregator Private Key
self-contained
```

```
EligibilityToken contents:
* iss – issuer identifier
* aud – audience
* exp – expiration
* iat – date-time of creation
* jti – unique id of JWT
* scope – scope for OperatorToken
* client_id – 3rd Party app client id
* callback_url – 3rd Party app <b>callbackUrl</b>
end note

server <- aggregator: 200 OK {“followUrl”:
“https://mno.example.com/?\neligibility={EligibilityToken}”}
server -> server: processAggregatorResponse()

rnote over server
3rd party app server derives data from EligibilityToken to identify session
end note

app <- server: 302 redirect
https://mno.example.com/?\neligibility={EligibilityToken}
app -> client: https://mno.example.com/?\neligibility={EligibilityToken}

rnote over client, es
TS.43 client authentication

For OperatorToken flow:
* TS.43 client has to be authenticated, i.e. initial authentication has to be
completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows
completed, at least once
* Eligibility check is always triggered
* EligibilityToken contains Aggregator details
end note

client -> client: process request from 3rd party app

note left: TS.43 client gets values from Eligibility Token: scope, client_id, aud,
callback_url

rnote over app, aggregator

Eligibility Token Consumption & Operator Token Generation

end note
rnote over app, aggregator

Operator Token Consumption

end note

@enduml
```



Figure 1: OperatorToken Flow with TS.43 Client - Generic - Eligibility Token Generation

[1] 3rd party app sends a request to 3rd party app server for Operator Token Scope, e.g. Number Verify

[2] 3rd party server sends a request to aggregator for Eligibility Token, that is required for a specific Operator Token scope, e.g. Number Verify

Aggregator prerequisites: • expose jwks endpoint • has to provision SP to MNOs • SP provisioning process (out of scope) (reference: numbering ranges announcement process)

[3] Aggregator validates request from 3rd party app server; checks 3rd party app onboarding details, e.g. scope, client_id, redirect_uri

Optionally, Aggregator calls MNO to get JWT This could be MNO from "circle of trust", that hosts 3rd party app details (whitelist of scopes, client_ids, redirect urls, etc.) and aggregator configuration

[4] Aggregator creates Eligibility Token that enables Operator Token Processing on a devices

Eligibility Token as a JWS – signed with Aggregator Private Key (self-contained)

The content of Eligibility Token is described in section 4.4.

[5] Aggregator responds to 3rd party app server with a 200 OK and provides a body in a response, that contains follow URL, that has to be processed by 3rd party app server. The content of Aggregator Response body is described in section 4.8., e.g.:

```
{
    "followUrl": "https://mno.example.com/?eligibility={EligibilityToken}"
}
```

[6] 3rd party app server process response from Aggregator

[7] 3rd party app server responds to 3rd party app with a 302 redirect to a location to TS.43 client, passing Eligibility Token:, e.g.:

```
302 redirect https://mno.example.com/?eligibility={EligibilityToken}
```

The content of redirect path is described in section 4.9

[8] 3rd party app calls TS.43 Client Operator Token API: https://mno.example.com/?eligibility={EligibilityToken}

TS.43 client authentication For Operator Token flow (pre-conditions):

- TS.43 client has to be authenticated, i.e. initial authentication has to be completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows completed, at least once
- Eligibility check is always triggered within a flow with ECS
- EligibilityToken contains Aggregator details

If TS.43 client authentication fails, 3rd Party app cannot complete Operator Token flow with TS.43 client. Client should return a respective error for this scenario.

[9] TS.43 client process request from 3rd Party app

TS.43 client derives following data from EligibilityToken:

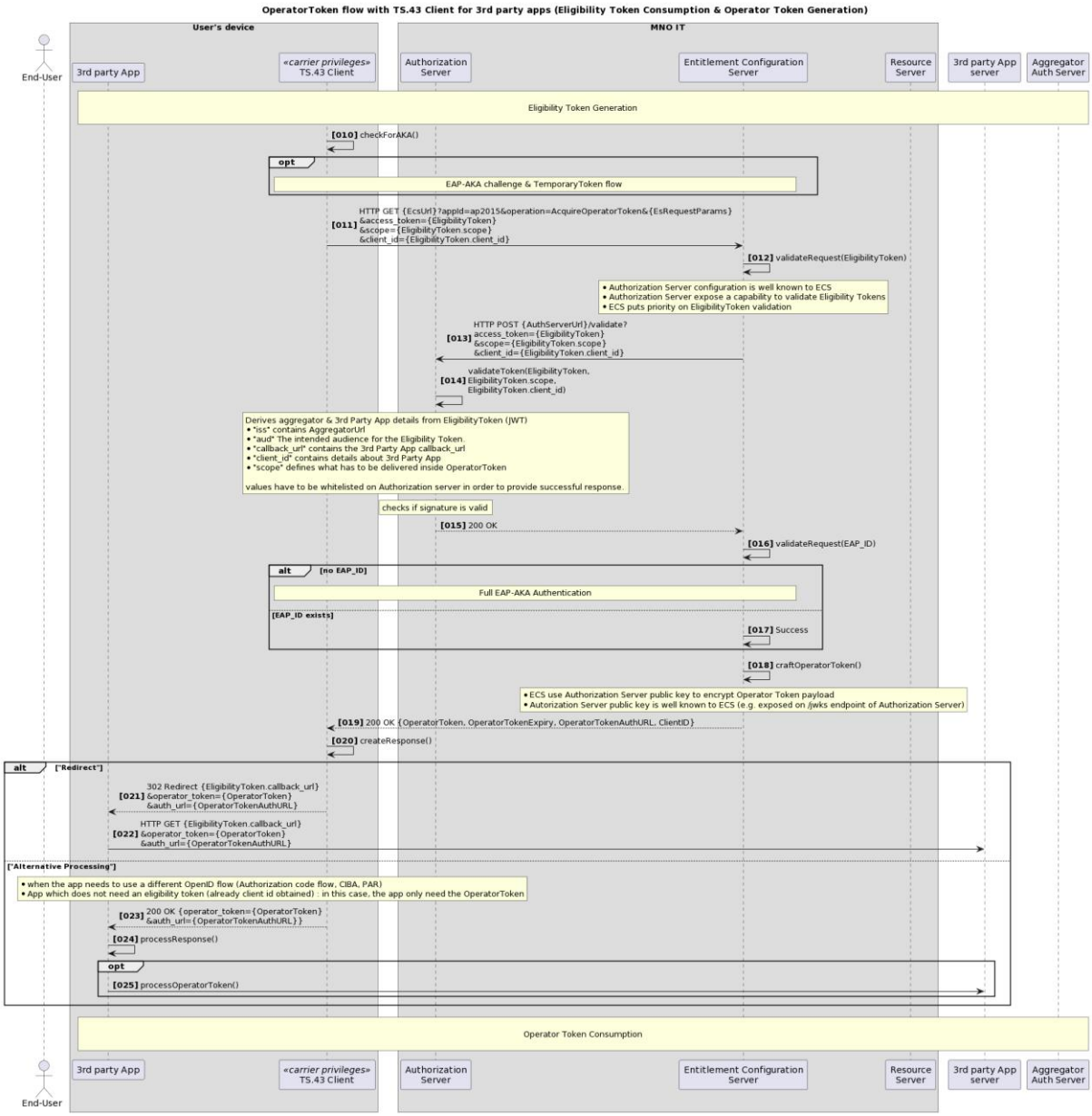
- Scope
- Client_id
- Aud (contains 3rd party app server url for handling redirectsaudience details that are intended to process Eligibility Token, i.e. ECS)
- Callback_url – contains 3rd party app server url for handling redirects

4.2 Eligibility Token consumption & Operator Token generation

```
@startuml OperatorToken Flow with TS.43 Client – Generic – Eligibility Token
Consumption & Operator Token Generation

title OperatorToken flow with TS.43 Client for 3rd party apps (Eligibility Token
Consumption & Operator Token Generation)



!pragma teoz true

autonumber 10 "<b>[000]"

actor "End-User" as user

box "User's device"
    participant "3rd party App" as app
    participant "TS.43 Client" as client <<carrier privileges>>
end box

box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box

participant "3rd party App\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

rnote over app, aggregator

Eligibility Token Generation
```

```
end note

client -> client: checkForAKA()

opt

rnote over client, es
EAP-AKA challenge & TemporaryToken flow
end note

end opt

client -> es: HTTP GET
{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}\n&access_tok
en={EligibilityToken}\n&scope={EligibilityToken.scope}\n&client_id={EligibilityToke
n.client_id}
es -> es: validateRequest(EligibilityToken)

rnote over es
* Authorization Server configuration is well known to ECS
* Authorization Server expose a capability to validate Eligibility Tokens
* ECS puts priority on EligibilityToken validation
end note

es -> idgw: HTTP POST
{AuthServerUrl}/validate?\naccess_token={EligibilityToken}\n&scope={EligibilityToke
n.scope}\n&client_id={EligibilityToken.client_id}
idgw -> idgw:
validateToken(EligibilityToken,\nEligibilityToken.scope,\nEligibilityToken.client_i
d)

rnote over idgw
Derives aggregator & 3rd Party App details from EligibilityToken (JWT)
* "iss" contains AggregatorUrl
* "aud" The intended audience for the Eligibility Token.
* "callback_url" contains the 3rd Party App callback_url
* "client_id" contains details about 3rd Party App
* "scope" defines what has to be delivered inside OperatorToken

values have to be whitelisted on Authorization server in order to provide
successful response.
End note

rnote over idgw
checks if signature is valid
end note

es ← idgw: 200 OK
es -> es: validateRequest(EAP_ID)

alt no EAP_ID
rnote over client, es
Full EAP-AKA Authentication
end note
else  EAP_ID exists
es -> es: Success
end alt

es -> es: craftOperatorToken()

rnote over es
* ECS use Authorization Server public key to encrypt Operator Token payload
* Autorization Server public key is well known to ECS (e.g. exposed on /jwks
endpoint of Authorization Server)
end note
```

```
client ← es: 200 OK {OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL, ClientID}
client -> client: createResponse()

alt "Redirect"

app ← client: 302 Redirect
{EligibilityToken.callback_url}\n&operator_token={OperatorToken}\n&auth_url={OperatorTokenAuthURL}
app -> server: HTTP GET
{EligibilityToken.callback_url}\n&operator_token={OperatorToken}\n&auth_url={OperatorTokenAuthURL}

else "Alternative Processing"

rnote over app, idgw
* when the app needs to use a different OpenID flow (Authorization code flow, CIBA, PAR)
* App which does not need an eligibility token (already client id obtained) : in this case, the app only need the OperatorToken
end note

app ← client: 200 OK
{operator_token={OperatorToken}\n&auth_url={OperatorTokenAuthURL}}
app -> app: processResponse()

opt

app -> server: processOperatorToken()

end opt

end alt

rnote over app, aggregator

Operator Token Consumption

end note
@enduml
```



[10] TS.43 Client checks for AKA on a device

Optional: TS.43 Client performs EAP-AKA challenge and invokes ECS to create Temporary Token

[11] TS.43 Client sends a request to ECS to create Operator Token (based on TS.43 specification):

HTTP GET
{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}&access_token={EligibilityToken}&scope={EligibilityToken.scope}&client_id={EligibilityToken.client_id}

[12] ECS validates requests – Eligibility Token

Authorization Server configuration is well known to ECS Authorization Server expose a capability to validate Eligibility Token ECS MUST validate the Eligibility Token before any other operation.

Authorization Server validation process requirements are described in 5.1.1 of this document.

[13] ECS requests Eligibility Token validation by Authorization Server and sends a request to validation endpoint

HTTP POST
{AuthServerUrl}/validate?access_token={EligibilityToken}&scope={EligibilityToken.scope}&client_id={EligibilityToken.client_id}

[14] Authorization Server validates Eligibility Token

Derives aggregator & 3rd Party App details from Eligibility Token (JWT) • "iss" contains AggregatorUrl • "aud" (contains audience details that are intended to process Eligibility Token, i.e. ECS) • Callback_url – contains 3rd party app server url for handling redirectscontains callbackUrl of 3rd Party App, as it is intended to consume OperatorToken content • client_id contains details about 3rd Party App • scope defines what hast to be delivered inside OperatorToken payload values have to be whitelisted on Authorization server in order to provide successful response. Authorization server checks signature of JWT

[15] Authorization Server provides a response to ECS about Eligibility token validation result (200 OK)

[16] ECS validates request – EAP_ID

Alternative 1 – if ECS does not recognize EAP_ID then ECS performs Full EAP-AKA authentication

Alternative 2 – ECS recognizes EAP_IP – go to the next steps

[17] Successful EAP_ID validation

[18] ECS process validation response and crafts Operator Token upon successful validation

ECS use Authorization Server public key to encrypt Operator Token payload Authorization Server public key is well known to ECS (e.g. exposed on /jwks endpoint of Authorization Server)

[19] ECS provides response (200 OK) to TS.43 client with Operator Token fields, i.e.: OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL, ClientID

[20] TS.43 Client creates a response to 3rd Party app

Alternative 1– redirect based response

[21] TS.43 Client sends 302 Redirect

```
302 redirect
{EligibilityToken.callback_url}?operator_token={OperatorToken}&auth_url={OperatorTokenAuthUrl}
```

The content of redirect path is described in section 4.10

[22] 3rd Party App follows location from redirect and continues the flow with 3rd Party App Server

Alternative 2 – alternative processing

[23] TS.43 Client sends 200 OK response and pass OperatorToken in response body.

```
200 OK
{EligibilityToken.callback_url}?operator_token={OperatorToken}&operator_token_expiry={OperatorTokenExpiry}&auth_url={OperatorTokenAuthUrl}
```

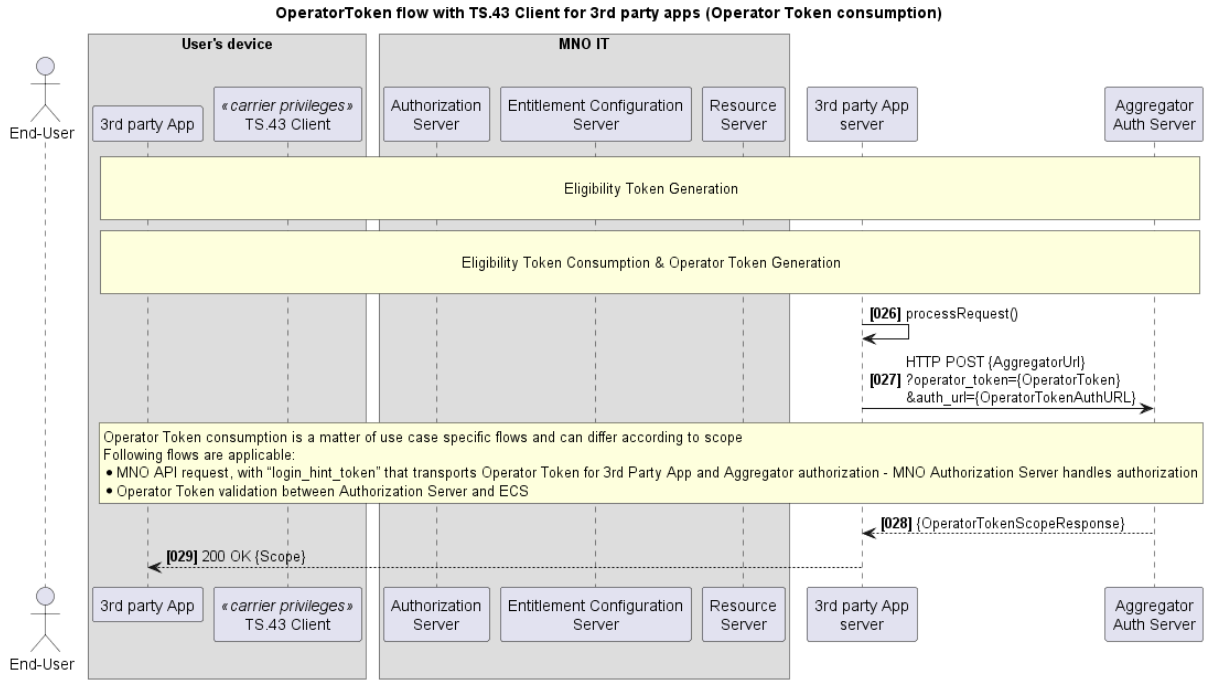
[24] 3rd Party App process response

[25] 3rd Party App sends back response to 3rd Party app server

4.3 Operator Token consumption

```
@startuml OperatorToken Flow with TS.43 Client – Generic – Operator Token
consumption

title OperatorToken flow with TS.43 Client for 3rd party apps (Operator Token
consumption)
!pragma teoz true

autonumber 26 "<b>[000]"

actor "End-User" as user

box "User's device"
    participant "3rd party App" as app
    participant "TS.43 Client" as client <<carrier privileges>>
end box

box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box

participant "3rd party App\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

rnote over app, aggregator

Eligibility Token Generation

end note
rnote over app, aggregator

Eligibility Token Consumption & Operator Token Generation

end note

server -> server: processRequest()
server -> aggregator: HTTP POST
{AggregatorUrl}\n?operator_token={OperatorToken}\n&auth_url={OperatorTokenAuthURL}

rnote over app , aggregator
Operator Token consumption is a matter of use case specific flows and can differ
according to scope
Following flows are applicable:
* MNO API request, with "login_hint_token" that transports Operator Token for 3rd
Party App and Aggregator authorization – MNO Authorization Server handles
authorization
* Operator Token validation between Authorization Server and ECS
end note

server ← aggregator: {OperatorTokenScopeResponse}
app ← server: 200 OK {Scope}
```

```
@enduml
```



[26] 3rd Party App server validates request from the app

[27] 3rd Party App server continues Operator Token flow with Aggregator

> Operator Token consumption is a matter of use case specific flows and can differ according to scope
>
> Following flows are applicable:
>
> - MNO API request, with "login_hint_token" that transports Operator Token for 3rd Party App and Aggregator authorization – MNO Authorization Server handles authorization
> - Operator Token validation between Authorization Server and ECS

[28] Aggregator provides a response (200 OK) to 3rd Party App Server according to desired scope

[29] 3rd Party App Server fulfils app request for Operator Token scope

4.4 Eligibility Token Payload (JWT)

Aggregator provides an Eligibility Token that follows RFC 7519 [6]:

Parameter	Required Category in Profile	Description
iss	REQUIRED	Issuer Identifier – a case-sensitive HTTPS based URL, with the host. It MAY contain the port and path element (OPTIONAL) but no query parameters. (AggretagorUrl)
aud	REQUIRED	The intended audience for the Eligibility Token. It is an array of case-sensitive strings. It MUST contain the ECS & TS.43 Client indicator.
exp	REQUIRED	The expiration time after which the Eligibility Token MUST NOT be accepted for processing. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
iat	REQUIRED	The time at which the Eligibility Token JWT was issued. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
jti	REQUIRED	The "jti" (JWT ID) claim provides a unique identifier for the JWT.The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string.
scope	REQUIRED	The value of the scope claim is a JSON string containing a space-separated list of scopes associated with the token, in the format described in Section 3.3 of [RFC6749].
client_id	REQUIRED	The client_id claim carries the client identifier of the Oauth 2.0 [RFC6749] client that requested the token.
callback_url	REQUIRED	It is an array of case-sensitive strings. It MUST contain the 3rd Party App callback_url, which is entitled to consume Operator Token (whitelisted on MNO side)
response_type	REQUIRED	A value of response_type parameter triggers an action to TS.43 Client to create a response to 3rd Party App according to configuration allowed by Eligibility Token issuer. It can be one of values: 1. "Redirection" or "302" – TS.43 client indicates that 3rd Party App must take additional action – redirect – in order to complete Operator Token request. Location of the redirect is captured from Eligibility

Parameter	Required Category in Profile	Description
		Token callback_url parameter. Operator Token is passed in response (query parameter), so 3rd Party App client can complete the flow on a given location. 2. "Success" or "200"– TS.43 client indicates that 3rd Party App request has been accepted/completed successfully and returns Operator Token in a response.
`cnf`	OPTIONAL	**The "cnf" confirmation claim value MUST represent only a single proof-of-possession key. the public key information is represented using the jkt confirmation method member defined in RFC 9449 (https://datatracker.ietf.org/doc/html/rfc9449#dpop_jkt)**

Eligibility Token is signed with Aggregator Private Key and can be processed only by the MNO Authorization Server.

4.5 Operator Token

Operators token to allow authentication for a 3rd party application on the device that may not have the means to acquire token or temporary_token

4.6 Operator Token scopes

Scope Parameter Value	Service	Description
`mc_vm_match`	Mobile Connect Verified MSISDN Match (Number Verify)	According to IDY.54 Mobile Connect Verified MSISDN Definition and Technical Requirements. Indicates that ECS SHALL add MSISDN inside Operator Token payload. MSISDN SHALL be encrypted by ECS with MNO Authorization Server Public Key.
`Si2:processes`	Signature service	Scope for operational use cases
`si2:registration`	Signature service	Scope for registration to the signature service
`si2:recovery`	Signature service	Scope for recovery uses cases
`si2:simsls`	Signature service	Scope for sim secret issuance
`si2:vcsbackup`	Signature service	Scope for credentials backup

Scope Parameter Value	Service	Description
`opw:subinf`	OpenGateway scope	Scope for Open Gateway use case. It is used for the GetSubscriberDeviceInfo operation in TS.43

NOTE: this list is not exhaustive. Further use cases can come in future versions of this specification.

4.7 Operator Token Mapping

For consumption of the Operator Token, the ECS must validate that the Scope used in obtaining the Operator Token has privileges for the operation used in the consumption. In the table below is the mapping of TS.43 **Operator Token Consumption** with the Scopes define in **Operator Token scopes**

Operation	Scope
ValidateOperatorToken	All Scopes
GetSubscriberDeviceInfo	opw:subinf, si2:processes, si2:registration, si2:recovery, si2:simsls, si2:vcsbackup

4.8 Aggregator Response – step 5

Aggregator sends back a response to 3rd party app server that contains Eligibility Token as a part of followUrl, which is coming in JSON in response body:

Key	Value	Description
`followUrl`	`String`	Contains TS.43 Client location. This can be e.g. an app scheme (AppLink[1]/UniversalLink[2]) or an URL that registered by the TS.43 client. This value has to be processed by 3rd party app and returned to a client to a device, in a form of 302 Redirect, to trigger Eligibility Token Consumption & Operator Token generation.

Security considerations

URL schemes are vulnerable to man-in-the-middle attack. Those attacks are described by PKCE[11]. It is more secure to associate a domain with the TS.43 client.

[1] Android App Links – https://developer.android.com/training/app-links

[2] iOS Universal Links – https://developer.apple.com/ios/universal-links/

4.9 3rd Party App Server Response – step 7

3rd Party App Server crafts 302 Redirect response to 3rd party app client on a device that contains followUrl from Aggregator response (see section 4.8) and additional query parameters:

Parameter	Usage Category	Description
eligibility	REQUIRED	The "ELIGIBILITY" is used to transport a Eligibility Token produced by Aggregator

4.10 TS.43 Client 302 Redirect Response parameters – step 21

TS.43 client crafts a 302 redirect towards CallbackUrl, derived from Eligibility Token "callback_url" key. Additionally, following query parameters are passed:

Parameter	Usage Category	Description
operator_token	REQUIRED	Operator Token, with a scope requested by the 3rd Party App
auth_url	REQUIRED	Operator Token Authorization Url – a location where Operator Token can be consumed. Usually, it is authorization endpoint of MNO Authorization Server. Only authorized entities can consume Operator Token (authentication & authorization concept, e.g. Mobile Connect OIDC or CIBA).

4.11 TS.43 Client 200 Response parameters – step 23

TS.43 client crafts a 200 response to 3rd party app, with following object in response body:

Parameter	Usage Category	Description
operator_token	REQUIRED	Operator Token, with a scope requested by the 3rd Party App
auth_url	REQUIRED	Operator Token Authorization Url – a location where Operator Token can be consumed. Usually, it is authorization endpoint of MNO Authorization Server. Only authorized entities can consume Operator Token (authentication & authorization concept, e.g. Mobile Connect OIDC or CIBA).
expiry_time	REQUIRED	Operator Token Expiry value. Indicates the time the provided OperatorToken expires

5 Interfaces characteristics

5.1 Authorization Server and Entitlement Configuration Server interface

Authorization Server enables an API for consumption to validate Eligibility Token

5.1.1 Authorization Server High level requirements

Requirement	Description
ASAC_RQ1.1 API exposure	The Authorization Server SHALL expose a common API towards Entitlement Configuration Server.
ASAC_RQ1.2 Endpoint exposure	Authorization Server SHALL expose "/validate" endpoint in the same domain as ECS, that allows ECS to validate Eligibility Token
ASAC_RQ1.3 Eligibility Token consumption	Authorization Server Validate API SHALL use query parameter "access_token" to pass Eligibility Token for validation
ASAC_RQ1.4 Eligibility validation result	Authorization Server SHALL return one of the following: • a positive result • a negative result with an appropriate error code
ASAC_RQ1.5 Security scheme	Authorization Server SHOULD enable "/validate" endpoint over standardized security scheme, e.g. OAUTH2.0 client credentials with a dedicated set of client_id and client_secret for ECS
ASAC_RQ1.6 High availability	The Authorization Server deployment SHOULD provide high availability/reliability.

5.2 3rd party application and TS.43 Client interface

TS.43 Client enables an API for 3rd party applications to get Operator Token

5.2.1 TS.43 Client High level requirements

Requirement	Description
ASAC_RQ2.1 OS support	TS.43 client SHOULD provide a support for all operating systems, e.g. iOS, Android, Windows, Linux
ASAC_RQ2.2 API exposure	TS.43 client SHALL expose a common API towards 3rd party applications
ASAC_RQ2.3 TS.43 Operator Token specification support	TS.43 client SHALL support Operator Token flow, as described in TS.43 specification, i.e. interact with ECS to get OperatorToken
ASAC_RQ2.4 Operator Token result	TS.43 client SHALL return one of the following: • a positive result with OperatorToken (200 OK) • a positive result with OperatorToken and redirect (302 Redirect) when Eligibility Token contains "EligibilityToken callback url • a negative result with an appropriate error code
ASAC_RQ2.5 Native API	TS.43 client SHOULD provide Operator Token interface towards 3rd party applications
ASAC_RQ2.6 multiSIM support	TS.43 client SHALL provide a support for multiSIM scenario and allow End-User to pick a SIM for OperatorToken flow

5.3 Authorization Server and Aggregator interface

Authorization Server enables an API to issue Eligibility Tokens & consume Operator Tokens

5.3.1 Authorization Server High level requirements

Requirement	Description
ASAC_RQ3.1 API exposure	Authorization Server deployment SHALL expose a common API towards Aggregator.
ASAC_RQ3.2 issue Eligibility Token	Authorization Server SHALL return Eligibility Token for OperatorToken flow on "/token" endpoint
ASAC_RQ3.3 Eligibility Token for 3rd party applications	The Authorization Server SHALL enable Eligibility Token for 3rd party applications that Aggregator onboards on Authorization Server
ASAC_RQ3.4 Eligibility Token request result	Authorization Server SHALL return one of the following: • a positive result (Eligibility Token) • a negative result with an appropriate error code
ASAC_RQ3.5 Eligibility Token content	Authorization Server SHALL return Eligibility Token as a JWT with a content described in 4.1
ASAC_RQ3.6 Operator Token consumption	Authorization Server SHALL allow OperatorToken consumption
ASAC_RQ3.6 Operator Token consumption – login_hint_token	Authorization Server SHALL allow Operator Token to be passed within OIDC "login_hint_token" parameter to capture authentication context from OperatorToken
ASAC_RQ3.7 Operator Token consumption	Authorization Server SHALL be capable of decrypting OperatorToken contents, including authentication context (ECS encrypts OperatorToken with Authorization Server public key), with Authorization Server private key.
ASAC_RQ3.8 Operator Token context consumption	Authorization Server SHALL follow Operator Token context, that defines use case, e.g. Mobile Connect Number Verify, Camara Authentication, eIDAS 2.0

6 Use cases Characteristics

6.1 Verifying a phone number provided by the user within an account registration flow (Mobile Connect Number Verify)

The Mobile Connect architecture consists of a Core framework [7] around which additional components can be added to support the different Mobile Connect services. The Core framework is based upon the OpenID Connect (OIDC) protocol suite [2] and allows Users to be identified by their MSISDN (or a related Pseudonymous Customer Reference) to enable authentication on their mobile device.

The serving Mobile Operator selects an appropriate Authenticator based on Mobile Operator policy, device capability and the Level of Assurance required by the SP to enable authentication.

Mobile Connect Number Verify (Verified MSISDN specification) allows SPs to verify the phone number of the device connected to the mobile data network through which a User is accessing a service provided by a SP.

TS.43 Operator Token enables SP to use Number Verify outside mobile data network context – when connected to Wi-Fi.

In order to assure continuous & seamless flow and fresh authentication data, it is recommended that:

1) Aggregator enables 3rd party application to consume Number Verify with Operator Token and use a common client_id for 3rd party application identified across MNO
2) Aggregator populates client_id to MNOs that enabled OperatorToken flow
3) Aggregator calls use "Eligibility Token" API from MNO to acquire JWT required for further processing of OperatorToken
4) Aggregator use HTTP 302 redirect to enable 3rd party application to utilize TS.43 Client Operator Token API
5) 3rd party application follows redirect from Aggregator and consumes TS.43 Client API for Operator Token, and pass Eligibility Token
6) TS.43 Client exposes API that consumes Eligibility token over a common interface, which is agnostic to operating system on a device
7) TS.43 Client use Eligibility Token as "access_token" on ECS OperatorToken interface
8) TS.43 Client responds with 302 redirect to Eligibility Token callback url and delivers Operator Token to 3rd party application backend to complete Number Verify flow with Aggregator

6.1.1 Eligibility Token Generation

```
@startuml OperatorToken Flow with TS.43 Client – Number Verify – Eligibility Token
Generation
title OperatorToken flow with TS.43 Client for 3rd party applications (Eligibility
Token Generation)

!pragma teoz true
autonumber "<b>[000]"

actor "End-User" as user
box "User's device"
    participant "3rd party Application" as application
    participant "TS.43 Client" as client <<carrier privileges>>
end box
box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box
participant "3rd party Application\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

application -> server: requestNumberVerify()
server -> aggregator: requestEligibilityForNumberVerify(SpDetails)
rnote over aggregator
Aggregator prerequisites:
```

```
* expose jwks endpoint
* has to to provision SP to MNOs
* SP provisioning process TBD
(reference: numbering ranges announcement process)
end note
aggregator -> aggregator: validateRequest(SpDetails)
rnote over aggregator
* aggregator calls MNO to get JWT (option)
* this could be a random MNO to build trust (MNO trust circle)
end rnote
aggregator -> aggregator: craftEligibilityToken()
rnote over aggregator
signed with Aggregator Private Key
self-contained
EligibilityToken contents:
* iss – issuer identifier
* aud – audience
* exp – expiration
* iat – date-time of creation
* jti – unique id of JWT
* scope – NumberVerify scope for OperatorToken
* client_id – 3rd Party application client id
* callback_url – 3rd Party application <b>callbackUrl</b>
end note
server <- aggregator: 200 OK {“followUrl”:
“https://mno.example.com/?\neligibility={EligibilityToken}”}
server -> server: processAggregatorResponse()
rnote over server
3rd party application server derives data from EligibilityToken to identify session
end note
application <- server: 302 redirect
https://mno.example.com/?\neligibility={EligibilityToken}
application -> client: https://mno.example.com/?\neligibility={EligibilityToken}
rnote over client, es
TS.43 client authentication

For OperatorToken flow:
* TS.43 client has to be authenticated, i.e. initial authentication has to be
completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows
completed, at least once
* Eligibility check is always triggered
* EligibilityToken contains Aggregator details
end note

client -> client: process request from 3rd party application
note left: TS.43 client gets values from Eligibility Token: scope, client_id, aud,
callback_url
rnote over application, aggregator

Eligibility Token Consumption & Operator Token Generation

end note
rnote over application, aggregator

Operator Token Consumption & Number Verify flow

end note
@enduml
```



[1] 3rd party application sends a request to 3rd party application server for Operator Token Scope, here Number Verify

[2] 3rd party server sends a request to aggregator for Eligibility Token, that is required for Number Verify

Aggregator prerequisites: • expose jwks endpoint • has to to provision SP to MNOs • SP provisioning process (out of scope) (reference: numbering ranges announcement process)

[3] Aggregator validates request from 3rd party application server; checks 3rd party application onboarding details, e.g. scope, client_id, redirect_uri

Optionally, Aggregator calls MNO to get JWT This could be MNO from "circle of trust", that hosts 3rd party application details (whitelist of scopes, client_ids, redirect urls, etc.) and aggregator configuration

[4] Aggregator creates Eligibility Token that enables Operator Token Processing on a devices

Eligibility Token as a JWS – signed with Aggregator Private Key (self-contained)

The content of Eligibility Token is described in section 4.4

[5] Aggregator responds to 3rd party application server with a 200 OK and provides a body in a response, that contains follow URL, that has to be processed by 3rd party application server. The content of Aggregator Response body is described in section 4.8., e.g.:

```
{
    "followUrl": "https://mno.example.com/?eligibility={EligibilityToken}

}
```

[6] 3rd party application server process response from Aggregator

[7] 3rd party application server responds to 3rd party application with a 302 redirect to a location to TS.43 client, passing Eligibility Token:, e.g.:

```
302 redirect https://mno.example.com/?eligibility={EligibilityToken}
```

The content of redirect path is described in section 4.9

[8] 3rd party application calls TS.43 Client Operator Token API:
https://mno.example.com/?eligibility={EligibilityToken}

TS.43 client authentication

For Operator Token flow (pre-conditions):

- TS.43 client has to be authenticated, i.e. initial authentication has to be completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows completed, at least once
- Eligibility check is always triggered within a flow with ECS
- EligibilityToken contains Aggregator details

If TS.43 client authentication fails, 3rd Party application cannot complete Operator Token flow with TS.43 client. Client should return a respective error for this scenario.

[9] TS.43 client process request from 3rd Party application

TS.43 client derives following data from EligibilityToken:

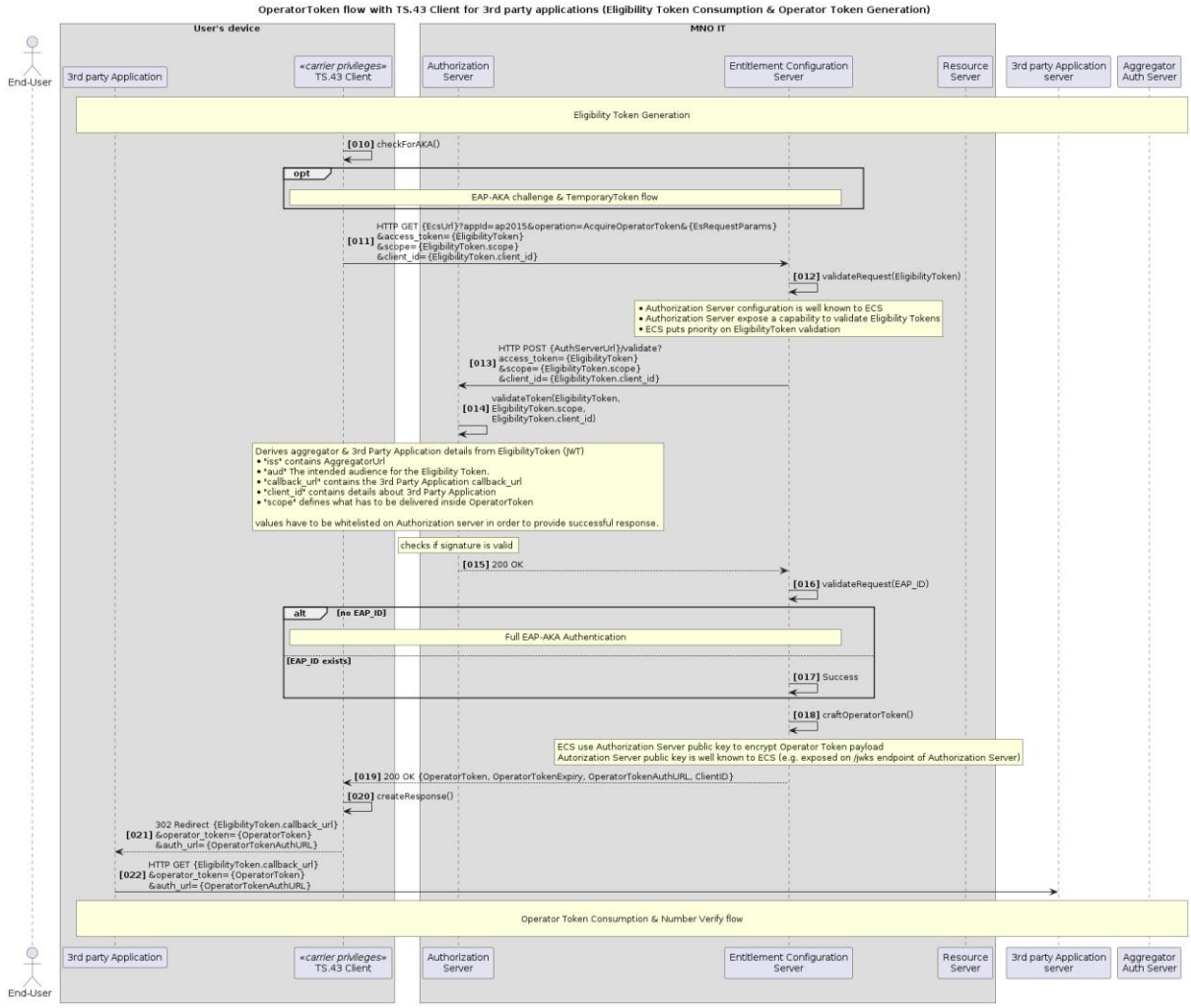
- Scope
- Client_id
- Aud (contains audience details that are intended to process Eligibility Token, i.e. ECS)
- Callback_url – contains 3rd party application server url for handling redirects

6.1.2 Eligibility Token consumption & Operator Token generation

```
@startuml OperatorToken Flow with TS.43 Client – Number Verify – Eligibility Token
Consumption & Operator Token Generation
title OperatorToken flow with TS.43 Client for 3rd party applications (Eligibility
Token Consumption & Operator Token Generation)

!pragma teoz true
autonumber 10 "<b>[000]"

actor "End-User" as user
box "User's device"
    participant "3rd party Application" as application
    participant "TS.43 Client" as client <<carrier privileges>>
end box
box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box
participant "3rd party Application\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

rnote over application, aggregator

Eligibility Token Generation

end note

client -> client: checkForAKA()
```

```
opt

rnote over client, es
EAP-AKA challenge & TemporaryToken flow
end note

end opt

client -> es: HTTP GET
{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}\n&access_tok
en={EligibilityToken}\n&scope={EligibilityToken.scope}\n&client_id={EligibilityToke
n.client_id}
es -> es: validateRequest(EligibilityToken)

rnote over es
* Authorization Server configuration is well known to ECS
* Authorization Server expose a capability to validate Eligibility Tokens
* ECS puts priority on EligibilityToken validation
end note

es -> idgw: HTTP POST
{AuthServerUrl}/validate?\naccess_token={EligibilityToken}\n&scope={EligibilityToke
n.scope}\n&client_id={EligibilityToken.client_id}
idgw -> idgw:
validateToken(EligibilityToken,\nEligibilityToken.scope,\nEligibilityToken.client_i
d)

rnote over idgw
Derives aggregator & 3rd Party Application details from EligibilityToken (JWT)
* "iss" contains AggregatorUrl
* "aud" The intended audience for the Eligibility Token.
* "callback_url" contains the 3rd Party Application callback_url
* "client_id" contains details about 3rd Party Application
* "scope" defines what has to be delivered inside OperatorToken

values have to be whitelisted on Authorization server in order to provide
successful response.
End note

rnote over idgw
checks if signature is valid
end note

es ← idgw: 200 OK
es -> es: validateRequest(EAP_ID)

alt no EAP_ID
rnote over client, es
Full EAP-AKA Authentication
end note
else  EAP_ID exists
es -> es: Success
end alt

es -> es: craftOperatorToken()
rnote over es
ECS use Authorization Server public key to encrypt Operator Token payload
Autorization Server public key is well known to ECS (e.g. exposed on /jwks endpoint
of Authorization Server)
end note
client ← es: 200 OK {OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL,
ClientID}
client -> client: createResponse()
application ← client: 302 Redirect
{EligibilityToken.callback_url}\n&operator_token={OperatorToken}\n&auth_url={Operat
orTokenAuthURL}
```

```
application -> server: HTTP GET
{EligibilityToken.callback_url}\n&operator_token={OperatorToken}\n&auth_url={Operat
orTokenAuthURL}
rnote over application, aggregator

Operator Token Consumption & Number Verify flow

end note
@enduml
```



[10] TS.43 Client checks for AKA on a device

Optional: TS.43 Client performs EAP-AKA challenge and invokes ECS to create Temporary Token

[11] TS.43 Client sends a request to ECS to create Operator Token (based on TS.43 specification):

```
HTTP GET
{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}\n
&access_token={EligibilityToken}\n&scope={EligibilityToken.scope}\n&client_id={El
igibilityToken.client_id}
```

[12] ECS validates requests – Eligibility Token

Authorization Server configuration is well known to ECS

Authorization Server expose a capability to validate Eligibility Tokens

ECS puts priority on EligibilityToken validation

Authorization Server validation process requirements are described in 5.1.1 of this document.

[13] ECS requests Eligibility Token validation by Authorization Server and sends a request to validation endpoint

```
HTTP POST
{AuthServerUrl}/validate?access_token={EligibilityToken}&scope={EligibilityToken.
scope}&client_id={EligibilityToken.client_id}
```

[14] Authorization Server validates Eligibility Token

Derives aggregator & 3rd Party Application details from Eligibility Token (JWT)

- "iss" contains AggregatorUrl
- "aud" (contains audience details that are intended to process Eligibility Token, i.e. ECS)
- Callback_url - contains 3rd party application server url for handling redirects
- client_id contains details about 3rd Party Application

- scope defines what hast to be delivered inside OperatorToken payload

values have to be whitelisted on Authorization server in order to provide successful response.

Authorization server checks signature of JWT

[15] Authorization Server provides a response to ECS about Eligibility token validation result (200 OK)

[16] ECS validates requests – EAP_ID

Alternative 1 – if ECS does not recognize EAP_ID then ECS performs Full EAP-AKA authentication

Alternative 2 – ECS recognizes EAP_IP – go to the next steps

[17] Successful EAP_ID validation

[18] ECS process validation response and crafts Operator Token upon successful validation

ECS use Authorization Server public key to encrypt Operator Token payload

Authorization Server public key is well known to ECS (e.g. exposed on /jwks endpoint of Authorization Server)

[19] ECS provides response (200 OK) to TS.43 client with Operator Token fields, i.e.: OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL, ClientID

[20] TS.43 Client creates a response to 3rd Party application

[21] TS.43 Client sends 302 Redirect

302 redirect {EligibilityToken.callback_url}?operator_token={OperatorToken}&auth_url={OperatorTokenAuthUrl}

The content of redirect path is described in section 4.10

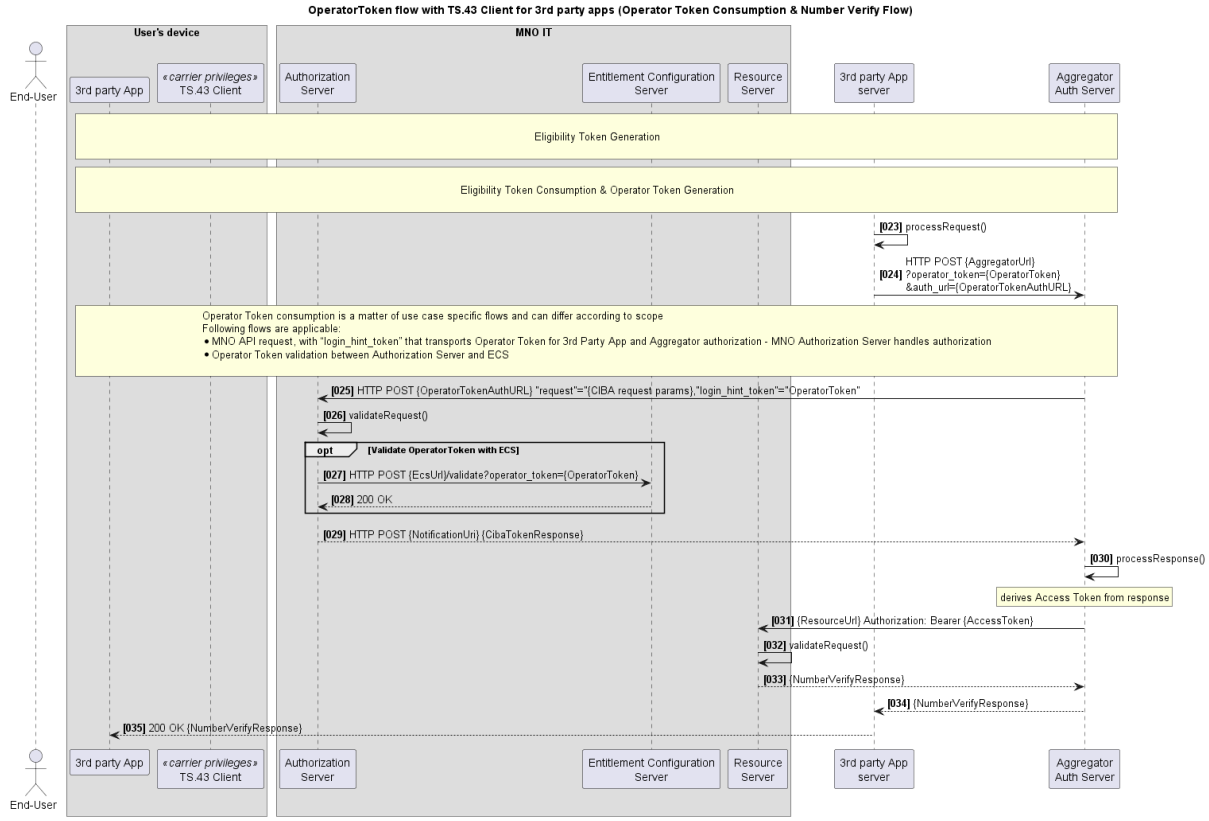
[22] 3rd Party Application follows location from redirect and continues the flow with 3rd Party Application Server (callback_url)

6.1.3 Operator Token consumption & Number Verify flow

```
@startuml OperatorToken Flow with TS.43 Client - Number Verify - Operator Token Consumption & Number Verify Flow
title OperatorToken flow with TS.43 Client for 3rd party applications (Operator Token Consumption & Number Verify Flow)

!pragma teoz true
autonumber 22 "<b>[000]"
```

```
actor "End-User" as user
box "User's device"
    participant "3rd party Application" as application
    participant "TS.43 Client" as client <<carrier privileges>>
end box
box "MNO IT"
    participant "Authorization\nServer" as idgw
    participant "Entitlement Configuration\nServer" as es
    participant "Resource\nServer" as resource
end box
participant "3rd party Application\nserver" as server
participant "Aggregator\nAuth Server" as aggregator

rnote over application, aggregator

Eligibility Token Generation

end note
rnote over application, aggregator

Eligibility Token Consumption & Operator Token Generation

end note

server -> server: processRequest()
server -> aggregator: HTTP POST
{AggregatorUrl}\n?operator_token={OperatorToken}\n&auth_url={OperatorTokenAuthURL}

rnote over application , aggregator
Operator Token consumption is a matter of use case specific flows and can differ
according to scope
Following flows are applicable:
* MNO API request, with “login_hint_token” that transports Operator Token for 3rd
Party Application and Aggregator authorization - MNO Authorization Server handles
authorization
* Operator Token validation between Authorization Server and ECS

end note

aggregator -> idgw: HTTP POST {OperatorTokenAuthURL} "request"="{CIBA request
params},"login_hint_token"="OperatorToken"
idgw -> idgw: validateRequest()
opt Validate OperatorToken with ECS
idgw -> es: HTTP POST {EcsUrl}/validate?operator_token={OperatorToken}
idgw <-- es: 200 OK
end opt
aggregator <-- idgw: HTTP POST {NotificationUri} {CibaTokenResponse}
aggregator -> aggregator: processResponse()
rnote over aggregator
derives Access Token from response
end note
aggregator -> resource: {ResourceUrl} Authorization: Bearer {AccessToken}
resource -> resource: validateRequest()
aggregator <-- resource:  {NumberVerifyResponse}
server <-- aggregator: {NumberVerifyResponse}
application <-- server: 200 OK {NumberVerifyResponse}
```

```
@enduml
```



[23] 3rd Party Application Server process response

[24] 3rd Party Application server continues Operator Token flow with Aggregator

> Operator Token consumption is a matter of use case specific flows and can differ according to scope
>
> Following flows are applicable:
>
> - MNO API request, with “login_hint_token” that transports Operator Token for 3rd Party Application and Aggregator authorization – MNO Authorization Server handles authorization
> - Operator Token validation between Authorization Server and ECS

[25] Aggregator sends authorization request to MNO AS, with *login_hint_token* request parameter that contains Operator Token, remaining query paramerters are described in IDY.01 [7]

[26] MNO AS validates request. Mobile Connect authorization request validation is described in IDY.04 [8]

ECS allows optional validation of Operator Token

[27] AS sends a request to ECS to validate Operator Token

[28] ECS provides a response (200OK) to AS upon successful Operator Token validation
[29] AS sends a response (e.g. notification) to Aggregator upon successful authorization step with Access Token to Number Verify resource
[30] Aggregator process authorization response and derives access token from it
[31] Aggregator sends a request to MNO Number Verify resource with access token derived from authorization response and remaining Number Verify parameters in a request to resource (subject to IDY.54 [9])
[32] MNO resource validates request
[33] AS sends a response to Aggregator with Number Verify result (Number Verify Response).
[34] Aggregator provides a Number Verify response (200 OK) to 3rd Party Application Server
[35] 3rd Party Application Server fulfils application request for Number Verify

6.2 MNO Native Application use case

An application may use device application authentication to retrieve an OperatorToken as a proof of a successful SIM authentication and use the OperatorToken as a login hint. In order to perform the SIM authentication, the device shall be connected to any data network (4G, 5G or WiFi).

In a Level of Assurance (LoA) High context, it is essential for the MNO native application to implement proof of possession as specified in [RFC9449] (https://datatracker.ietf.org/doc/html/rfc9449). The OperatorToken MUST be bound to the client's proof-of-possession key to implement an end-to-end binding of the entire authorization flow.

An application may not be authorized to use an aggregator e.g. out of policy, security or regulation reasons. In this case a dedicated backend service - Eligibility Service - contractually if not technically bound to the MNO must be responsible for delivering an Eligibility Token.

Finally to address uses cases where the Application does not know which backend to use as it depends on the particular MNO that issued the SIM (basically for any service that would be standard across MNOs) a discovery service is required.

One example of MNO native application use case can be for digital identity Wallet in the context of eIDAS [1].

MNO native application use case illustrates the requirements mentioned above.

In order to assure continuous & seamless flow and fresh authentication data, it is recommended that:

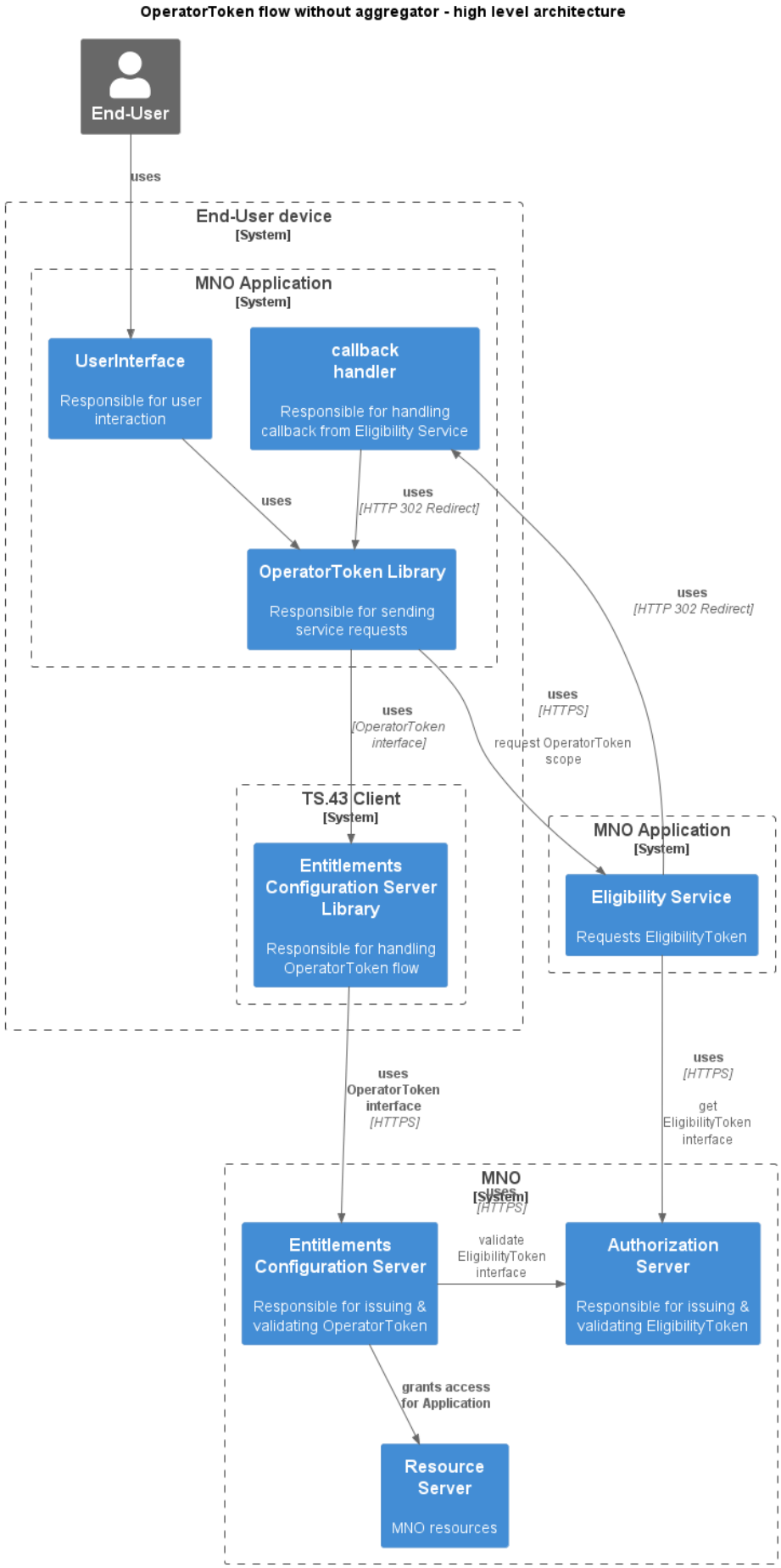
1) TS.43 Client exposes API that provide the Eligibility Service end-point over a common interface, which is agnostic to operating system on a device
2) Eligibility Service enables Application to consume services with OperatorToken and use a common client_id for application identified across MNO

3) Eligibility Service populates client_id to MNOs that enabled OperatorToken flow
4) Eligibility Service calls “Eligibility Token” API from MNO to acquire Eligibility Token JWT required for further processing of OperatorToken
5) Eligibility Service uses HTTP 302 redirect to enable application to use TS.43 Client Operator Token API
6) Application follows redirect from Eligibilty Service and consumes TS.43 Client API for Operator Token, and passes Eligibility Token
7) TS.43 Client exposes API that consumes Eligibility Token over a common interface, which is agnostic to operating system on a device
8) TS.43 Client uses Eligibility Token as “access_token” on ECS OperatorToken interface
9) TS.43 Client responds with 200 ok issuing an Operator Token that grants access to MNO services

6.2.1 High level architecture without aggregator

```
@startuml silentToken/OperatorToken flow without aggregator - high level
architecture
!include <c4/C4_Context.puml>
!include <c4/C4_Container.puml>
!include <c4/C4_Component.puml>
!include <office/Users/user.puml>
!include <office/Users/mobile_user.puml>

title OperatorToken flow without aggregator - high level architecture
HIDE_STEREOTYPE()

Person_Ext(user, "End-User")
System_Boundary(app, "MNO Application") {
    Container(appServer, "Eligibility Service", "", "Requests EligibilityToken")
}
System_Boundary(mno, "MNO") {
    Container(es, "Entitlements\nConfiguration Server", "", "Responsible for
issuing & validating OperatorToken")
    Container(authServ, "Authorization\nServer", "", "Responsible for issuing &
validating EligibilityToken")
    Container(resource, "Resource\nServer", "", "MNO resources")
}
System_Boundary(smartPhone, "End-User device") {
    System_Boundary(ts43app, "TS.43 Client") {
        Container(esLibrary, "Entitlements Configuration Server Library", "",
"Responsible for handling OperatorToken flow")
    }
    System_Boundary(appFront, "MNO Application") {
        Container(ui, "UserInterface", "", "Responsible for user interaction")
        Container(otLib, "OperatorToken Library", "", "Responsible for sending
service requests")
        Container(callback, "callback\nhandler", "", "Responsible for handling
callback from Eligibility Service")
    }
}
Rel(user, ui, "uses", "")
Rel(ui, otLib, "uses")
Rel(otLib, appServer, "uses", "HTTPS", "request OperatorToken scope")
Rel(callback, otLib, "uses","HTTP 302 Redirect")
Rel(otLib, esLibrary, "uses","OperatorToken\ninterface")
Rel(appServer,authServ,"uses","HTTPS","get\nEligibilityToken\ninterface","Eligibili
tyToken")
```

```
Rel(esLibrary,es, "uses\nOperatorToken\ninterface","HTTPS")
Rel_R(es, authServ, "uses","HTTPS","validate\nEligibilityToken\ninterface")
Rel_D(es,resource, "grants access\nfor Application")
Rel(appServer,callback,"uses","HTTP 302 Redirect")
Lay_R(es, authServ)
Lay_R(authServ, resource)
@enduml
```


OperatorToken flow without aggregator - high level architecture
End-User
uses
End-User device
[System]
MNO Application
[System]
UserInterface
Responsible for user interaction
callback handler
Responsible for handling callback from Eligibility Service
uses
uses
[HTTP 302 Redirect]
OperatorToken Library
Responsible for sending service requests
uses
[HTTP 302 Redirect]
uses
[HTTPS]
request OperatorToken scope
uses
[OperatorToken interface]
TS.43 Client
[System]
Entitlements Configuration Server Library
Responsible for handling OperatorToken flow
MNO Application
[System]
Eligibility Service
Requests EligibilityToken
uses
OperatorToken interface
[HTTPS]
uses
[HTTPS]
get EligibilityToken interface
MNO
[System]
uses
[HTTPS]
Entitlements Configuration Server
Responsible for issuing & validating OperatorToken
validate EligibilityToken interface
Authorization Server
Responsible for issuing & validating EligibilityToken
grants access for Application
Resource Server
MNO resources

6.2.2 MNO Native Application use case flow

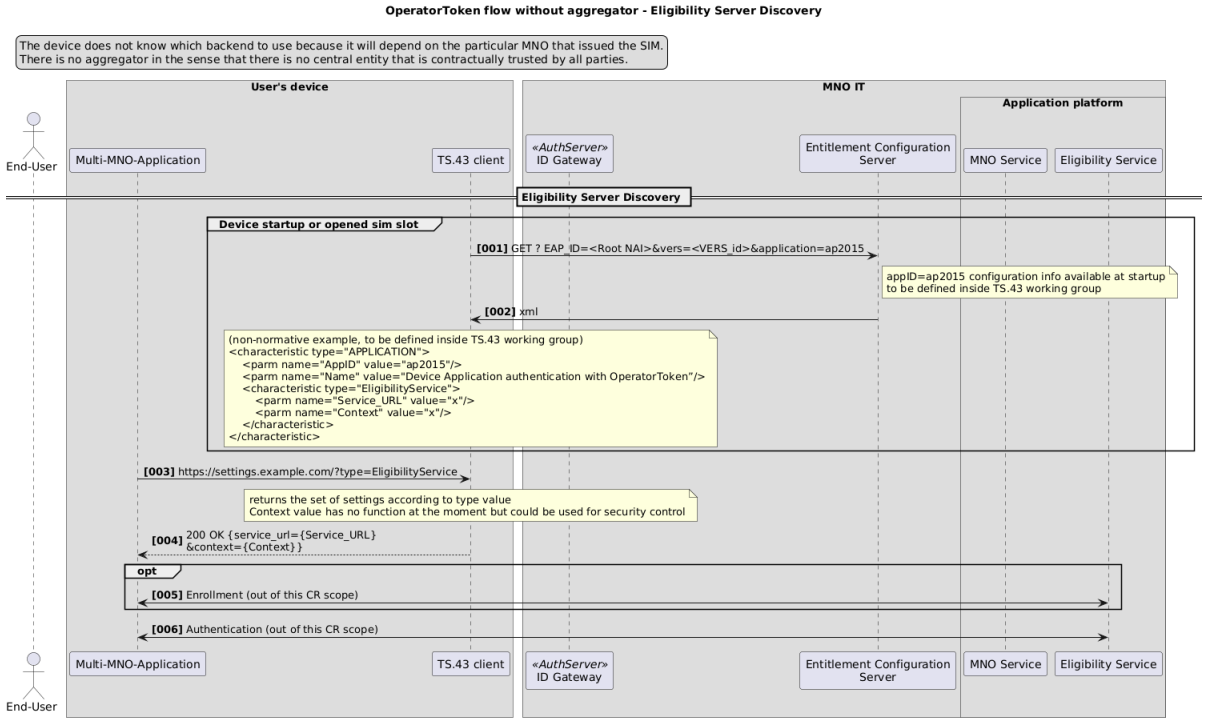
The flow is divided into 4 sections :

- Eligibility Service Discovery
- Eligibility Token Generation
- Operator Token Generation
- Operator Token Consumption

6.2.2.1 Eligibility Service Discovery

```
@startuml silentToken/OperatorTokenFlowWithoutAggregator - Discovery v1
title OperatorToken flow without aggregator - Eligibility Server Discovery

legend top left
  The device does not know which backend to use because it will depend on the
particular MNO that issued the SIM.
  There is no aggregator in the sense that there is no central entity that is
contractually trusted by all parties.
end legend

!pragma teoz true
autonumber "<b>[000]"

actor "End-User" as user
box "User's device" <<Android OS>>
    participant "Multi-MNO-Application" as application
    participant "TS.43 client" as client
end box
box "MNO IT"
    participant "ID Gateway" as idgw <<AuthServer>>
    participant "Entitlement Configuration\nServer" as es
    box "Application platform"
        participant "MNO Service" as resource
        participant "Eligibility Service" as server
    end box
end box

== Eligibility Server Discovery ==
group Device startup or opened sim slot
    client -> es : GET ? EAP_ID=<Root NAI>&vers=<VERS_id>&application=ap2015
    note right es
        appID=ap2015 configuration info available at startup
        to be defined inside TS.43 working group
    end note
    client <- es : xml
    note over client
        (non-normative example, to be defined inside TS.43 working group)
        <characteristic type="APPLICATION">
            <parm name="AppID" value="ap2015"/>
            <parm name="Name" value="Device Application authentication with
OperatorToken”/>
            <characteristic type="EligibilityService">
                <parm name="Service_URL" value="x"/>
                <parm name="Context" value="x"/>
            </characteristic>
        </characteristic>
    end note
end group

application -> client: https://settings.example.com/?type=EligibilityService
```

```
note over client
    returns the set of settings according to type value
    Context value has no function at the moment but could be used for security
control
end note
application <-- client: 200 OK {service_url={Service_URL}\n&context={Context}}
opt
    application <-> server : Enrollment (out of this CR scope)
end opt
application <-> server : Authentication (out of this CR scope)
@enduml
```



Device startup or opened sim slot

[001] TS.43 client retrieves Application Entitlement Settings for ap2015. Optionally, TS.43 Client may request Application Entitlement Settings for all Application in one request

configuration info available at startup to be defined inside TS.43 working group

[002] ECS returns Application Entitlement Setting for ap2015

(non-normative example, to be defined inside TS.43 working group)

```
<characteristic type="APPLICATION">
        <parm name="AppID" value="ap2015"/>
        <parm name="Name" value="Device Application authentication with
OperatorToken”/>
        <characteristic type="EligibilityService">
            <parm name="Service_URL" value="x"/>
            <parm name="Context" value="x"/>
        </characteristic>
```

</characteristic>

Runtime

[003] Multi-MNO-Application ask for the EligibilityTokenService URL

https://settings.example.com/?type=EligibilityService

[004] TS.43 Client returns the Service URL and a Context value

Context value has no function at the moment but could be used for security control

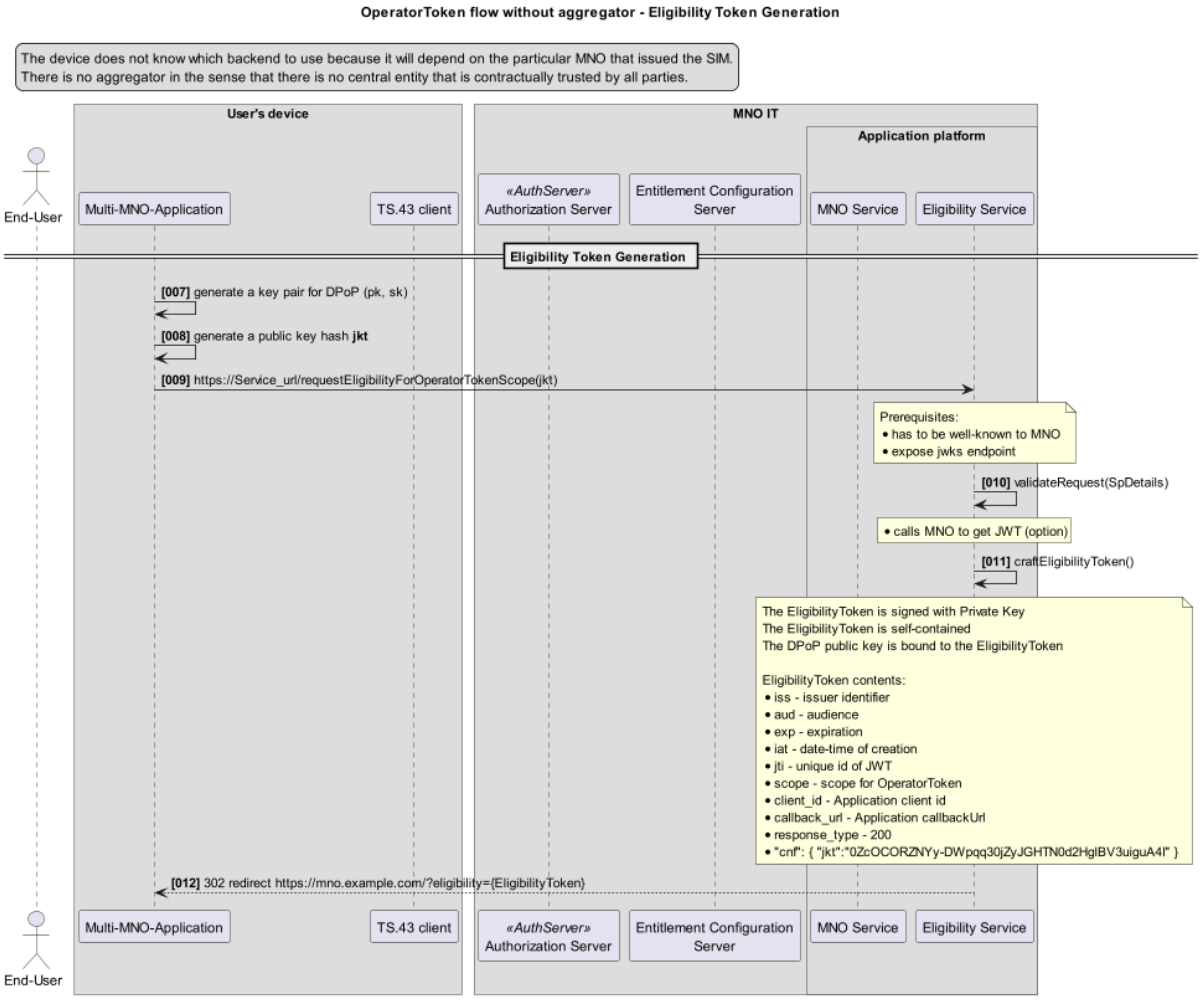
[005] End-user and/or Multi-MNO-Application must enroll to Eligibility Service backend (out of this CR scope)

[006] End-user and/or Multi-MNO-Application must authenticate to Eligibility Service backend (out of this CR scope)

6.2.2.2 Eligibility Token Generation

```
@startuml silentToken/OperatorTokenFlowWithoutAggregator -
EligibilityTokenGeneration v1.2
title OperatorToken flow without aggregator - Eligibility Token Generation

legend top left
  The device does not know which backend to use because it will depend on the
particular MNO that issued the SIM.
  There is no aggregator in the sense that there is no central entity that is
contractually trusted by all parties.
end legend

!pragma teoz true
autonumber 7 "<b>[000]"

actor "End-User" as user
box "User's device" <<Android OS>>
    participant "Multi-MNO-Application" as application
    participant "TS.43 client" as client
end box
box "MNO IT"
    participant "Authorization Server" as idgw <<AuthServer>>
    participant "Entitlement Configuration\nServer" as es
    box "Application platform"
        participant "MNO Service" as resource
        participant "Eligibility Service" as server
    end box
end box

== Eligibility Token Generation ==

application -> application : generate a key pair for DPoP (pk, sk)
application -> application : generate a public key hash **jkt**

application -> server :
https://Service_url/requestEligibilityForOperatorTokenScope(jkt)
note over server
    Prerequisites:
    * has to be well-known to MNO
    * expose jwks endpoint
end note
server -> server: validateRequest(SpDetails)
rnote over server : * calls MNO to get JWT (option)
server -> server: craftEligibilityToken()
```

```
note over server
    The EligibilityToken is signed with Private Key
    The EligibilityToken is self-contained
    The DPoP public key is bound to the EligibilityToken

    EligibilityToken contents:
    * iss - issuer identifier
    * aud - audience
    * exp - expiration
    * iat - date-time of creation
    * jti - unique id of JWT
    * scope - scope for OperatorToken
    * client_id - Application client id
    * callback_url - Application callbackUrl
    * response_type - 200
    * "cnf": { "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I" }
end note
application <-- server: 302 redirect
https://mno.example.com/?eligibility={EligibilityToken}
@enduml
```



[007] Multi-MNO-Application generates a key pair for proof-of-possession mechanism. The Multi-MNO-Application can use the OS TEE (Keystore / Secure Enclave) to increase the level of security.

[008] Multi-MNO-Application calculates the hash of the public key

[009] Multi-MNO-Application sends a request to the Eligibility Service for Operator Token Scope

Eligibility Server prerequisites: • has to be well-known to MNO • expose jwks endpoint

[010] Eligibility Server validates request from Eligibility Service; checks Multi-MNO-Application onboarding details, e.g. scope, client_id, redirect_uri

Eligibility Server calls MNO to get JWT (option)

[011] Eligibility Authorization Server creates Eligibility Token that enables Operator Token Processing on a device

Eligibility Token as a JWS – signed with Eligibility Authorization Server Private Key (self contained) The content of Eligibility Token is described in section 4.4

[012] Eligibility Service responds to Multi-MNO-Application with a 302 redirect to TS.43 client, passing Eligibility Token:, e.g.:

302 redirect https://mno.example.com/?eligibility={EligibilityToken}

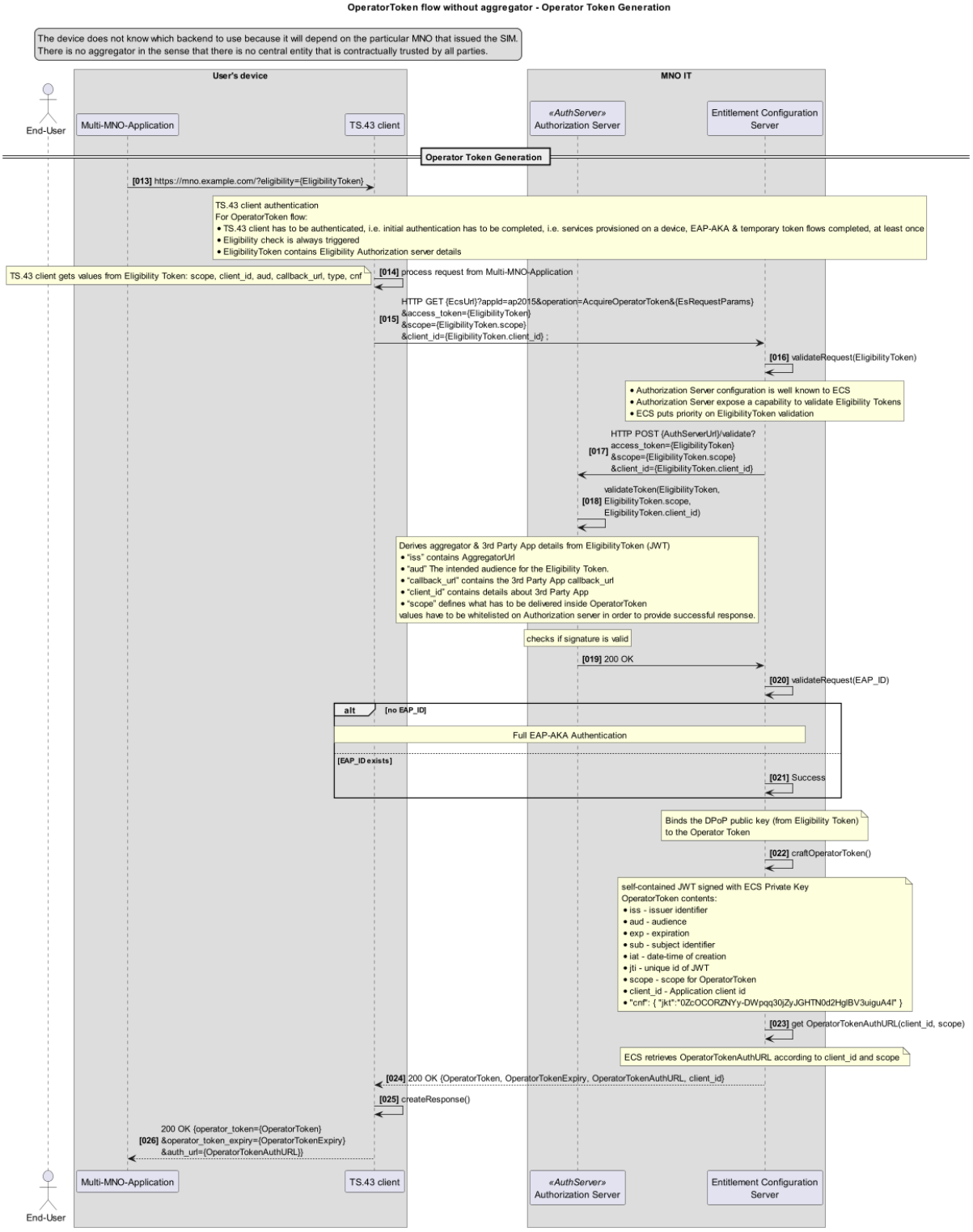
The content of redirect path is described in section 4.9

6.2.2.3 Operator Token Generation

```
@startuml silentToken/OperatorTokenFlowWithoutAggregator - OperatorTokenGeneration
v1
title OperatorToken flow without aggregator - Operator Token Generation

legend top left
  The device does not know which backend to use because it will depend on the
particular MNO that issued the SIM.
  There is no aggregator in the sense that there is no central entity that is
contractually trusted by all parties.
end legend

!pragma teoz true
autonumber 13 "<b>[000]"

actor "End-User" as user
box "User's device" <<Android OS>>
    participant "Multi-MNO-Application" as application
    participant "TS.43 client" as client
end box
box "MNO IT"
    participant "Authorization Server" as idgw <<AuthServer>>
    participant "Entitlement Configuration\nServer" as es
end box
```

```
== Operator Token Generation ==

application -> client: https://mno.example.com/?eligibility={EligibilityToken}
rnote over client, es
    TS.43 client authentication
    For OperatorToken flow:
    * TS.43 client has to be authenticated, i.e. initial authentication has to be
completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows
completed, at least once
    * Eligibility check is always triggered
    * EligibilityToken contains Eligibility Authorization server details
end note

client -> client: process request from Multi-MNO-Application
note left: TS.43 client gets values from Eligibility Token: scope, client_id, aud,
callback_url, type, cnf


client -> es: HTTP GET
{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}\n&access_tok
en={EligibilityToken}\n&scope={EligibilityToken.scope}\n&client_id={EligibilityToke
n.client_id} ;
es -> es: validateRequest(EligibilityToken)

rnote over es
    * Authorization Server configuration is well known to ECS
    * Authorization Server expose a capability to validate Eligibility Tokens
    * ECS puts priority on EligibilityToken validation
end note

es -> idgw: HTTP POST
{AuthServerUrl}/validate?\naccess_token={EligibilityToken}\n&scope={EligibilityToke
n.scope}\n&client_id={EligibilityToken.client_id}
idgw -> idgw:
validateToken(EligibilityToken,\nEligibilityToken.scope,\nEligibilityToken.client_i
d)



rnote over idgw
    Derives aggregator & 3rd Party App details from EligibilityToken (JWT)
    * "iss" contains AggregatorUrl
    * "aud" The intended audience for the Eligibility Token.
    * "callback_url" contains the 3rd Party App callback_url
    * "client_id" contains details about 3rd Party App
    * "scope" defines what has to be delivered inside OperatorToken
    values have to be whitelisted on Authorization server in order to provide
successful response.
End note

rnote over idgw
    checks if signature is valid
end note

es <- idgw: 200 OK
es -> es: validateRequest(EAP_ID)

alt no EAP_ID
    rnote over client, es
        Full EAP-AKA Authentication
    end note
else EAP_ID exists
    es -> es: Success
end alt

note over es
 Binds the DPoP public key (from Eligibility Token)
```

```
 to the Operator Token
end note

es -> es: craftOperatorToken()
note over es
    self-contained JWT signed with ECS Private Key
    OperatorToken contents:
    * iss - issuer identifier
    * aud - audience
    * exp - expiration
    * sub - subject identifier
    * iat - date-time of creation
    * jti - unique id of JWT
    * scope - scope for OperatorToken
    * client_id - Application client id
    * "cnf": { "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I" }
end note
es -> es : get OperatorTokenAuthURL(client_id, scope)
note over es : ECS retrieves OperatorTokenAuthURL according to client_id and scope
client <-- es: 200 OK {OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL,
client_id}
client -> client: createResponse()
application <-- client: 200 OK
{operator_token={OperatorToken}\n&operator_token_expiry={OperatorTokenExpiry}\n&aut
h_url={OperatorTokenAuthURL}}
@enduml
```



[013] Multi-MNO-Application calls TS.43 Client Operator Token API:

https://mno.example.com/?eligibility={EligibilityToken}

TS.43 client authentication For Operator Token flow:

- TS.43 client has to be authenticated, i.e. initial authentication has to be completed, i.e. services provisioned on a device, EAP-AKA & temporary token flows completed, at least once
- Eligibility check is always triggered
- EligibilityToken contains Eligibility Authorization Server details

[014] TS.43 client processes request from Multi-MNO-Application

[015] TS.43 Client sends a request to ECS to create Operator Token (based on TS.43 specification):

HTTP GET

{EcsUrl}?appId=ap2015&operation=AcquireOperatorToken&{EsRequestParams}\n&access_token={EligibilityToken}\n&scope={EligibilityToken.scope}\n&client_id={EligibilityToken.client_id}

[016] ECS validates requests – Eligibility Token

[017] ECS requests Eligibility Token validation by Authorization Server and sends a request to validation endpoint

[018] Authorization Server validates Eligibility Token

[019] Authorization Server provides a response to ECS about Eligibility token validation result (200 OK)

[020] ECS validates requests – EAP_ID

Alternative 1 – if ECS does not recognize EAP_ID then ECS performs Full EAP-AKA authentication

Alternative 2 – ECS recognizes EAP_IP – go to the next steps

[021] Successful EAP_ID validation

[022] ECS crafts OperatorToken

self-contained JWT signed with ECS Private Key

OperatorToken contents:

* iss - issuer identifier
* aud - audience
* exp - expiration
* sub - subject identifier
* iat - date-time of creation
* jti - unique id of JWT
* scope - scope for OperatorToken
* client_id - Application client id
* cnf: DPoP key binding

OperatorToken JWT signed with ECS Private Key ECS MUST exposed on /jwks endpoint

[023] ECS retrieves OperatorTokenAuthURL according to client_Id and scope

[024] ECS provides response (200 OK) to TS.43 client with Operator Token fields, i.e.: OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL, ClientID

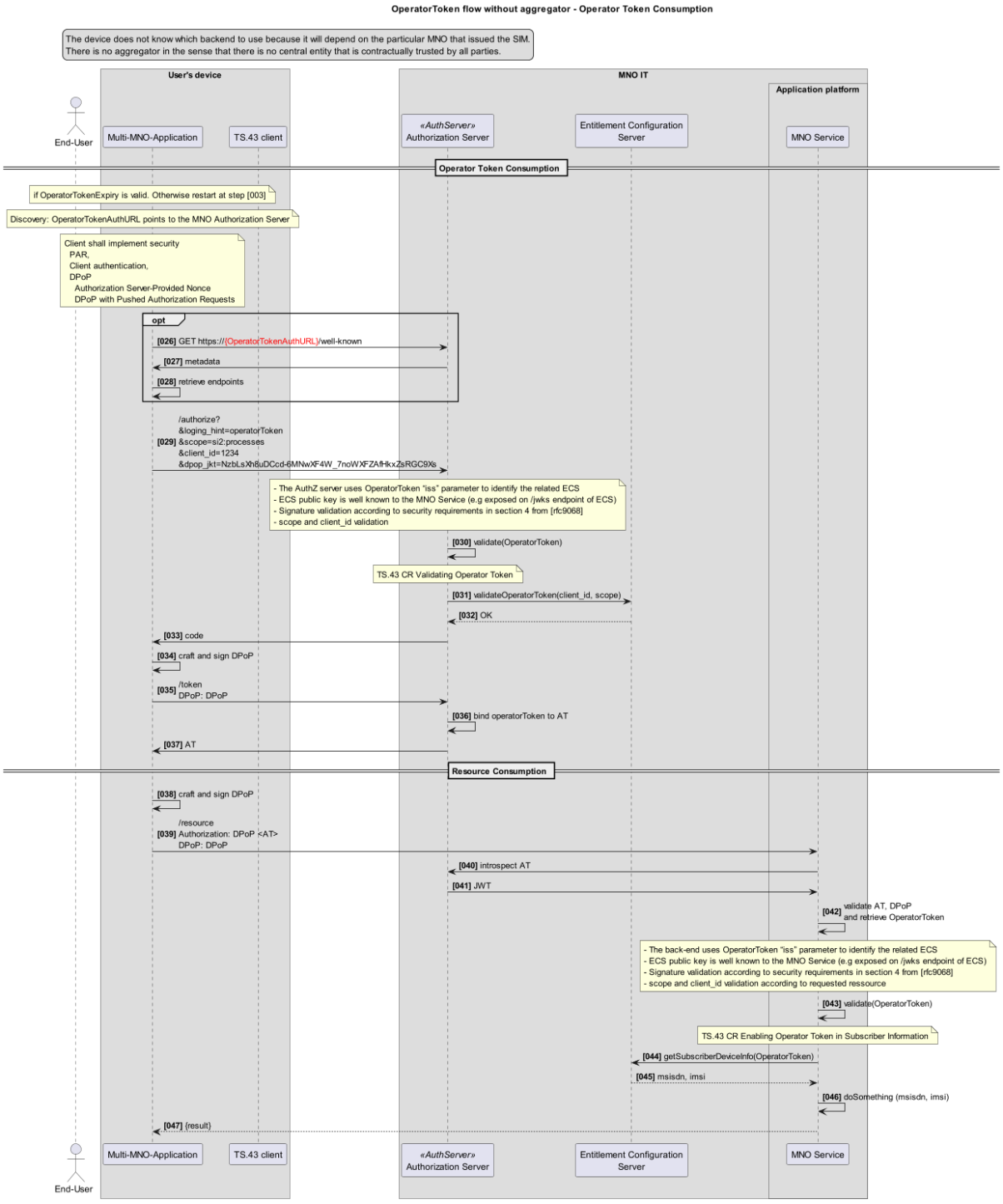
[025] TS.43 Client creates a response to Multi-MNO-Application

[026] TS.43 Client sends 200 OK response and passes OperatorToken, OperatorTokenExpiry, OperatorTokenAuthURL in response body.

6.2.2.4 Operator Token Consumption

```
@startuml silentToken/OperatorTokenFlowWithoutAggregator - OperatorTokenConsumption
v1

title OperatorToken flow without aggregator - Operator Token Consumption

legend top left
    The device does not know which backend to use because it will depend on the
particular MNO that issued the SIM.
    There is no aggregator in the sense that there is no central entity that is
contractually trusted by all parties.
end legend

!pragma teoz true
autonumber 26 "<b>[000]"

actor "End-User" as user
box "User's device" <<Android OS>>
    participant "Multi-MNO-Application" as application
    participant "TS.43 client" as client
end box
box "MNO IT"
    participant "Authorization Server" as idgw <<AuthServer>>
    participant "Entitlement Configuration\nServer" as es
    box "Application platform"
        participant "MNO Service" as resource
    end box
end box

== Operator Token Consumption ==
note over application : if OperatorTokenExpiry is valid. Otherwise restart at step
[003]
note over application
    Discovery: OperatorTokenAuthURL points to the MNO Authorization Server
end note

note over application
  Client shall implement security
    PAR,
    Client authentication,
    DPoP
      Authorization Server-Provided Nonce
      DPoP with Pushed Authorization Requests
end note
```

```
opt
application -> idgw : GET https://<color red>{OperatorTokenAuthURL}</color>/well-
known
application <- idgw : metadata
application -> application : retrieve endpoints
end opt

application -> idgw :
/authorize?\n&loging_hint=operatorToken\n&scope=si2:processes\n&client_id=1234
\n&dpop_jkt=NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs


note over idgw
- The AuthZ server uses OperatorToken "iss" parameter to identify the related ECS
- ECS public key is well known to the MNO Service (e.g exposed on /jwks endpoint of
ECS)
- Signature validation according to security requirements in section 4 from
[rfc9068]
- scope and client_id validation
end note
idgw -> idgw : validate(OperatorToken)
note over idgw : TS.43 CR Validating Operator Token
idgw -> es : validateOperatorToken(client_id, scope)
idgw <-- es : OK
application <- idgw : code

application -> application : craft and sign DPoP
application -> idgw : /token \nDPoP: DPoP
idgw -> idgw : bind operatorToken to AT
application <- idgw : AT

== Resource Consumption ==

application -> application : craft and sign DPoP
application -> resource : /resource \nAuthorization: DPoP <AT> \nDPoP: DPoP

resource -> idgw : introspect AT
resource <- idgw : JWT

resource -> resource : validate AT, DPoP \nand retrieve OperatorToken

note over resource
- The back-end uses OperatorToken "iss" parameter to identify the related ECS
- ECS public key is well known to the MNO Service (e.g exposed on /jwks endpoint of
ECS)
- Signature validation according to security requirements in section 4 from
[rfc9068]
- scope and client_id validation according to requested ressource
end note
resource -> resource : validate(OperatorToken)

note over resource : TS.43 CR Enabling Operator Token in Subscriber Information
resource -> es : getSubscriberDeviceInfo(OperatorToken)
resource <-- es : msisdn, imsi
resource -> resource : doSomething (msisdn, imsi)
application <-- resource : {result}
@enduml
```



Discovery: OperatorTokenAuthURL points to the MNO Authorization Server.

[026] The Multi-MNO-Application requests the well-known endpoint of the Authorization Server

[027] The Multi-MNO-Application gets the Authorization Server metadata

[028] The Multi-MNO-Application retrieves from the metadata the resources endpoints

[029-36] The Multi-MNO-Application starts an authorization code flow demontrating proof-of-possession. The Multi-MNO-Application can also implement PAR, client authentication and handle Nonce.

https:// OperatorTokenAuthURL /authorize? &loging_hint=operatorToken &scope=si2:processes &client_id=1234 &dpop_jkt=NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs

[037] The Multi-MNO-Application gets a Sender-Constrained Access Token

[038-41] The Multi-MNO-Application requests a resource demontrating proof-of-possession

[042] The Authorization Server validates the Access Token and retrieves the OperatorToken (token introspection)

[043] The MNO Service backend validates the OperatorToken

- The back-end uses OperatorToken “iss” parameter to identify the related ECS - ECS public key is well known to the MNO Service (e.g exposed on /jwks endpoint of ECS) - Signature validation according to security requirements in section 4 from [rfc9068] - Scope and client_id validation according to requested ressource

[044] The MNO Service asks to the identified ECS subscriber information

[045] ECS returns device msisdn ans imsi

[046] The MNO Service continues his process using msisdn and imsi

[047] The MNO Service returns the requested resource

6.2.3 TS.43 client URI-scheme – step [003]

URI : https://settings.example.com/

Parameter	Required Category in Profile	Description
`type`	REQUIRED	Setting identifier of parameters to be retrieved

Application request using a identifier of the parameters to be retrieved

https://settings.example.com/type=EligibilityService

TS.43 Client responds the parameters name and value identified by the request type parameter

```
<characteristic type="EligibilityService">
        <parm name="Service_URL" value="x"/>
        <parm name="Context" value="x"/>
</characteristic>
```

Note: mechanism how TS.43 Client will retrieve Application Entitlement Settings for ap2015 (steps [001] and [002]) will be described in TS.43 v12 document

6.2.4 OperatorToken (JWT) – step [013]

Self contained OperatorToken JWT follows RFC 9068:

Parameter	Required Category in Profile	Description
`iss`	REQUIRED	Issuer Identifier - a case-sensitive HTTPS based URL, with the host. It MAY contain the port and path element (OPTIONAL) but no query parameters. (ECSUrl)
`aud`	REQUIRED	The intended audience for the Operator Token. It is an array of case-sensitive strings. MUST use a default resource indicator.
`exp`	REQUIRED	The expiration time after which the Operator Token MUST NOT be accepted for processing. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
`sub`	REQUIRED	The "sub" (subject) claim identifies the principal that is the subject of the JWT. The claims in a JWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique. The processing of this claim is generally application specific. The "sub" value is a case-sensitive string containing a StringOrURI value
`iat`	REQUIRED	The time at which the Operator Token JWT was issued. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
`jti`	REQUIRED	The "jti" (JWT ID) claim provides a unique identifier for the JWT.The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string.

Parameter	Required Category in Profile	Description
scope	REQUIRED	The value of the scope claim is a JSON string containing a space-separated list of scopes associated with the token, in the format described in Section 3.3 of [RFC6749].
client_id	REQUIRED	The client_id claim carries the client identifier of the OAuth 2.0 [RFC6749] client that requested the token.
cnf	OPTIONAL	The "cnf" confirmation claim value MUST represent only a single proof-of-possession key. the public key information is represented using the jkt confirmation method member defined in RFC 9449 (https://datatracker.ietf.org/doc/html/rfc9449#dpop_jkt)

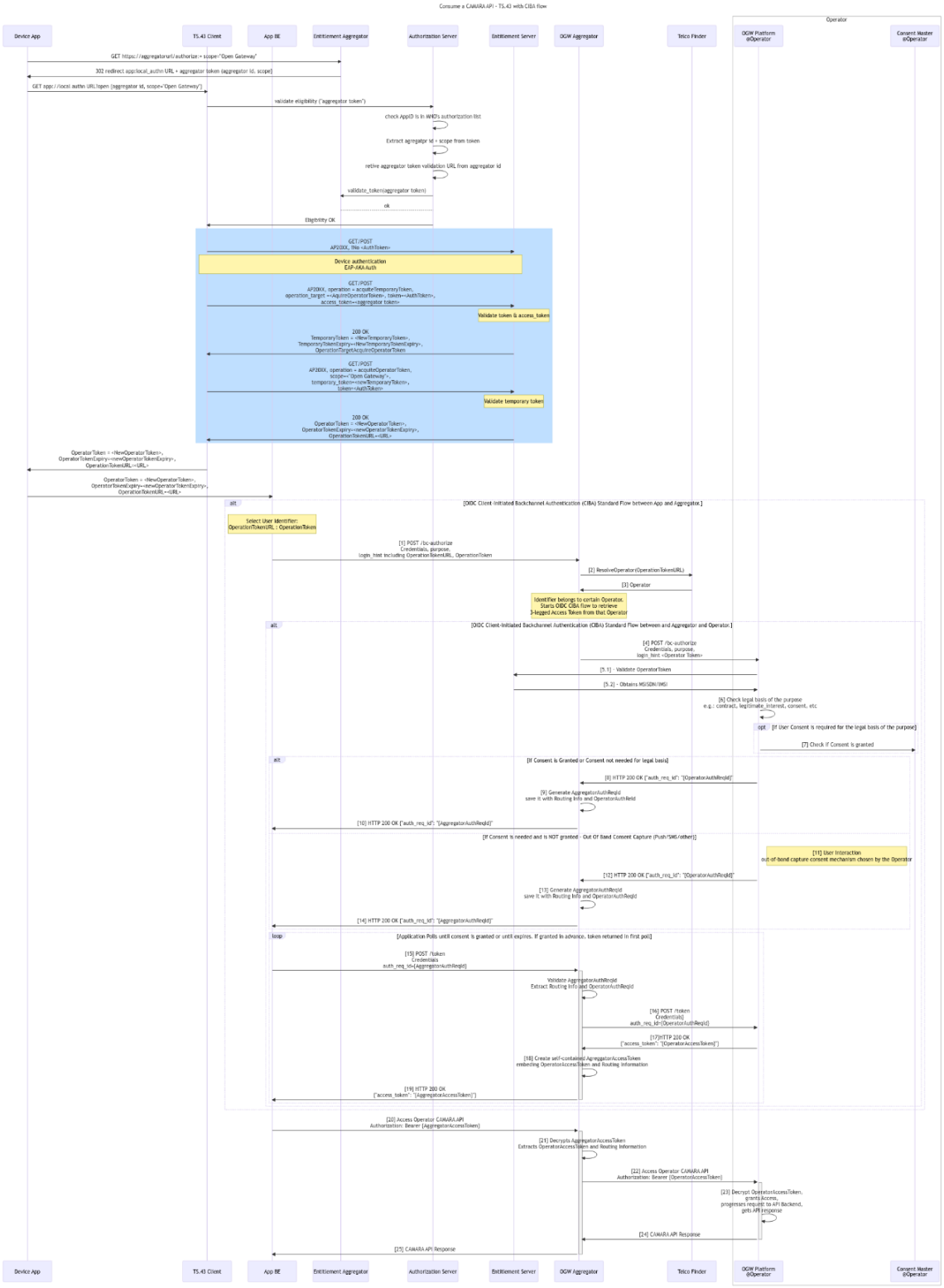
6.2.5 Entitlement Configuration Server High level requirements

Requirement	Description
ASAC_RQ4.1 OperatorToken as JWT	The Entitlement Configuration Server SHALL be able to issue an OperatorToken following RCF 9068 as defined in section 6.2.4. The OperatorToken SHALL be signed and complies with the JWS standard (JSON Web Signature, RFC 7515)
ASAC_RQ4.2 Endpoint exposure	Entitlement Configuration Server SHALL expose “/jwks” end-point that returns the Entitlement Configuration Server's public key set in the JWKS format
ASAC_RQ4.3 Operation availability	Entitlement Configuration Server SHALL implement “getSubscriberDeviceInfo” operation as specificied in section 14.6.2 of TS.43 v11 document
ASAC_RQ4.4 High availability	The Entitlement Configuration Server deployment SHOULD provide high availability/reliability.

6.3 Open Gateway use case

6.3.1 Overview

The following flow exemplifies how to use the ASAC procedures for an Open Gateway use case. In such case, the Operator Token is utilized to present the identity of the subscriber (MSISDN/IMSI) in order to generate an Access Token for third party applications to consume Network APIs from Open Gateway. This mechanism intents therefore to substitute the usage of the User Equipment IP address as a login hint in the current Open Gateway procedures.

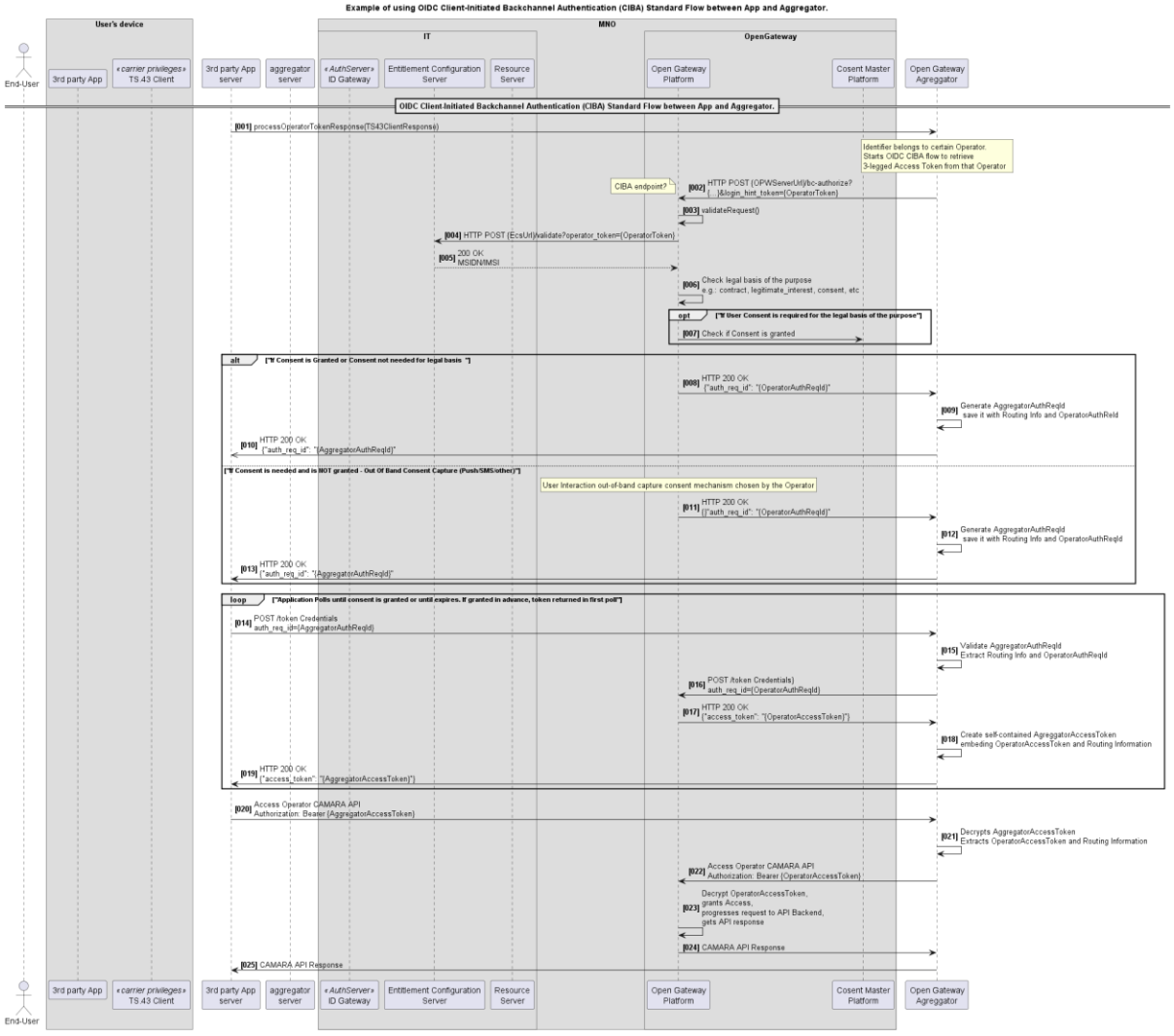


In order to accommodate the Open Gateway use case and other use cases, the ASAC procedures shall define in a separate manner the Operator Token generation from the Operator token consumption.

- Operator token generation apparently looks common across different use cases and flows.
- Operator token consumption can be different depending of the use case. the flow for the consumption therefore would normally be pre-existing flows (e.g. open gateway flow)

The Authorization server in the generation phase is a separate functional entity from the consumption case. The latter, in this example is within the Open Gateway platform of the MNO.

6.3.2 Interface flow (sequence diagram) – AccessToken consumption



Flow Description: (Note the flow is an example)

[1] The third-Application server contacts Open Gateway Aggregator for the consumption of an API.
[2] The Open Gateway Aggregator starts a OIDC CIBA Flow against Open Gateway platform.

[3] Open Gateway Platform will validate internally the request. That all the Parameters are correct.
[4] Open Gateway Platform will contact the ECS to validate the Operator Token received.
[5] ECS will validate the Operator Token and return the MSISDN and/or IMSI of the client that made the Operator Token Petition.
[6] The Open Gateway will validate the legal bases of the petition.
[7] Optional, if consent is needed. It will check with the Consent Master platform.

If consent is not needed the platform will generate the authorization Request ID for the process. [8][9][10]

If consent is needed

Out of band method to capture the consent. The platform will generate the authorization Request ID for the process. [11][12][13]

Pooling Method of CIBA. [14][15][16][17][18][19]

[20] The 3rd Party Server, with the Token from Open gateway Aggregator, makes a call to an CAMARA API.
[21] Open gateway Aggregator check the parameters and token passed.
[22] The Open Gateway aggregator contacts the Open Gateway platform with the token and Camara API that it requests.
[23] Open Gateway checks the validity of the token, and it process the API request.
[24] The API responds to the Aggregator.
[25] The Aggregator relay the response to the 3rd Party Server.

6.3.3 Functional requirements

1. The Authorization server used during the Operator token generation for the application eligibility check can be a separate entity from the Authorization server used in the consumption case (as shown in the above example of the Open Gateway use case).
2. In such case, during the consumption phase, the ECS must provide an interface towards the AuthServer in Open gateway to validate the Operator token and provide the corresponding MSISDN/IMSI.

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
ASAC.01 v1.0	26 April 2024	CR0001R06 Initial Flows	ISAG	Yolanda Sanz, GSMA
		CR0002R02 Example flow for the open gateway use case CR0003R05 Adding content to document CR0004R05 Example flow and requirements for a MNO Native Application use case CR0005R02 Scope Operation Mapping		
		First quality check: 1) Fixing references 2) Sections placement		
		Fix app by application		
		Fix all the editorial comments received during the Quality Review		
ASAC.01 v1.1	04 February 2025	Create Draft 0 of ASAC.01 v1.1 based on ASAC.01 v1.0	ISAG	Yolanda Sanz, GSMA
		CR11001R01 Fixing issues		
		CR11002R01 relaxes the format of followUrl.		
		CR11003R02 Fixing typos across the entire document		
ASAC.01 v1.2	31 March 2025	CR12001 Sender Constrain OperatorToken	ASAC	Yolanda Sanz, GSMA

A.2 Other Information

Type	Description
Document Owner	ASACG
Editor / Company	Yolanda Sanz, GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.