|  |  |
| --- | --- |
| 3GPP TR 33.884 V0.4.0 (2023-01) | |
| Technical Report | |
| 3rd Generation Partnership Project;  Technical Specification Group Services and System Aspects;  Study on security of application enablement aspects for subscriber-aware northbound API access  (FS\_SNAAPPY)  (Release 18) | |
|  | |
|  | 3GPP-logo_web |
|  | |
| The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP. The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented. This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification. Specifications and Reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices. | |

|  |
| --- |
|  |
| ***3GPP***  Postal address  3GPP support office address  650 Route des Lucioles - Sophia Antipolis  Valbonne - FRANCE  Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16  Internet  http://www.3gpp.org |
| ***Copyright Notification***  No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.  © 2022, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).  All rights reserved.  UMTS™ is a Trade Mark of ETSI registered for the benefit of its members  3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  GSM® and the GSM logo are registered and owned by the GSM Association |

Contents

Foreword 5

1 Scope 7

2 References 7

3 Definitions of terms, symbols and abbreviations 8

3.1 Terms 8

3.2 Symbols 8

3.3 Abbreviations 8

4 Assumptions 8

4.1 Architectural assumptions 8

5 Key issues 9

5.1 Key issue #1: Checking authentication and authorization of invoker 9

5.1.1 Key issue details 9

5.1.3 Potential security requirements 9

5.2 Key Issue #2: Checking authorization before allowing access 9

5.2.1 Key issue details 9

5.2.3 Potential security requirements 9

5.X Key issue #X: <Title> 10

5.X.1 Key issue details 10

5.X.2 Threats 10

5.X.3 Potential security requirements 10

6 Proposed solutions 10

6.0 Mapping of solutions to key issues 10

6.1 Solution #1: Resource Owner Authorization in API Invocation using OAuth Token 10

6.1.1 Introduction 10

6.1.2 Solution details 11

6.1.2.1 Architecture 11

6.1.2.2 Procedure 12

6.1.2.3 OAuth 2.0 role mapping 13

6.1.3 Evaluation 14

6.2 Solution #2: Authentication using OpenID Connect 14

6.2.1 Introduction 14

6.2.2 Solution details 15

6.2.3 Evaluation 16

6.3 Solution #3: UE Originated API invocation using OAuth Client Credential Grant 16

6.3.1 Introduction 16

6.3.2 Solution details 16

6.3.3 Evaluation 18

6.4 Solution #4: Authenticate and authorize UE in UE originated API invocation 19

6.4.1 Introduction 19

6.4.2 Solution details 19

6.4.3 Evaluation 21

6.5 Solution #5: Resource Owner based authorization for resource access 21

6.5.1 Introduction 21

6.5.2 Solution details 22

6.5.3 Evaluation 23

6.6 Solution #6: Authorization before allowing access to resources 23

6.6.1 Introduction 23

6.6.2 Solution details 23

6.6.3 Evaluation 25

6.7 Solution #7: Authorizing UE originated API invocation with PKCE flow 26

6.7.1 Introduction 26

6.7.2 Solution details 26

6.7.3 Evaluation 27

6.8 Solution #8: Validation of OAuth Token 27

6.8.1 Introduction 27

6.8.2 Solution details 27

6.8.3 Evaluation 28

6.9 Solution #9: OAuth 2.0 based API invocation procedure 28

6.9.1 Introduction 28

6.9.2 Solution details 28

6.9.3 Evaluation 31

6.10 Solution #10: UE credential based API invocation procedure 31

6.10.1 Introduction 31

6.10.2 Solution details 31

6.10.3 Evaluation 33

6.11 Solution #11: Providing and Revoking Resource Owner Authorization using OAuth 2.0 Authorization Code Grant 33

6.11.1 Introduction 33

6.11.2 Solution details 34

6.11.2.1 Architecture 34

6.11.2.2 Procedure 34

6.11.2.3 S-KID 36

6.11.2.4 KSNAAPPY derivation function 37

6.11.3 Evaluation 37

6.12 Solution #12: Providing and Revoking Resource Owner Authorization 37

6.12.1 Introduction 37

6.12.2 Solution details 37

6.12.2.1 Architecture 37

6.12.2.2 Procedure 38

612.2.3 S-KID 40

6.12.2.4 KAuz derivation function 40

6.12.2.5 Verification information derivation 40

6.12.3 Evaluation 40

6.Y Solution #Y: <Title> 41

6.Y.1 Introduction 41

6.Y.2 Solution details 41

6.Y.3 Evaluation 41

Annex <X>: Change history 42

For definitive guidance on drafting 3GPP TSs and TRs, see [3GPP TS 21.801](http://www.3gpp.org/DynaReport/21801.htm) supplemented by the 3GPP web page <http://www.3gpp.org/specifications-groups/delegates-corner/writing-a-new-spec>.

Ensure all blue guidance text is removed before submitting the TS/TR to the TSG for approval.

# Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, modal verbs have the following meanings:

**shall** indicates a mandatory requirement to do something

**shall not** indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

**should** indicates a recommendation to do something

**should not** indicates a recommendation not to do something

**may** indicates permission to do something

**need not** indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

**can** indicates that something is possible

**cannot** indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

**will** indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**will not** indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**might** indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

**might not** indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

**is** (or any other verb in the indicative mood) indicates a statement of fact

**is not** (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

# 1 Scope

The scope of present document is based on the requirements for SNA (TS 22.261 clause 6.10.2) [2] and on the Study on application enablement aspects for subscriber-aware northbound API access (TR 23.700-95) [3].

The objective of this study is to:

1. Identify potential new security requirements related to API invocation (such as user authorization) and define potential solutions to fulfil these requirements. This encompasses:

- Whether and how CAPIF functions can determine the resource owner upon CAPIF invocation

- Whether and how CAPIF can support obtaining authorization from the resource owner

- Whether and how CAPIF can support revocation of authorization by the resource owner

- Whether and how CAPIF can support security procedures with the aim to reduce authorization inquiries for a nested API invocation

The study is not exclusively tailored to CAPIF, but should align with widely deployed authorization frameworks.

2. Identify potential security requirements for APIs used in SNAAPP and define potential solutions to fulfil these requirements.

This objective includes UE-originated API invocation.

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 22.261: "Service requirements for the 5G system".

[3] 3GPP TR 23.700-95: “Study on application enablement aspects for subscriber-aware northbound API access”.

[4] IETF RFC 6749: “The OAuth 2.0 Authorization Framework”.

[5] 3GPP TS 33.122: “Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs”.

[6] openID.net: " OpenID Connect Core 1.0 incorporating errata set 1". Available at: <https://openid.net/specs/openid-connect-core-1_0.html>

[7] IETF RFC 7009: “OAuth 2.0 Token Revocation”.

[8] IETF RFC 7515: “JSON Web Signature (JWS)”.

[9] IETF RFC 7636: "Proof Key for Code Exchange by OAuth Public Clients".

[10] IETF RFC 7662: " OAuth 2.0 Token Introspection".

[11] IETF RFC 7542: "The Network Access Identifier".

# 3 Definitions of terms, symbols and abbreviations

This clause and its three subclauses are mandatory. The contents shall be shown as "void" if the TS/TR does not define any terms, symbols, or abbreviations.

## 3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

Definition format (Normal)

**<defined term>:** <definition>.

**example:** text used to clarify abstract rules by applying them literally.

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

AEF Application Exposure Function

ANF authentication function

API Application Programming Interface

Authz Authorization

AZF authorization function

CAPIF Common API Framework for 3GPP northbound APIs

## 3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

Abbreviation format (EW)

<ABBREVIATION> <Expansion>

# 4 Assumptions

Editor's note: This clause will capture the assumptions for this work

## 4.1 Architectural assumptions

Editor's note: This clause will capture the architectural assumptions for this work

# 5 Key issues

## 5.1 Key issue #1: Checking authentication and authorization of invoker

### 5.1.1 Key issue details

Only certain invokers are permitted to invoke subscriber aware northbound APIs, Therefore it is necessary to authenticate and authorize these invokers to access the APIs. The requirements for CAPIF apply.

For SNAAPP, the API invoker can also reside on a UE.

### 5.1.3 Potential security requirements

The requirements for the CAPIF-2 interface of 33.122 clause 4.4 shall apply:

(quoted for convenience)

The CAPIF-2/2e reference points between the API invoker and API exposing function shall fulfil the following requirements:

- [CAPIF-SEC-4.4-a] Mutual authentication between the API invoker and the API exposing function shall be supported.

- [CAPIF-SEC-4.4-b] The transport of messages over the CAPIF-2 and CAPIF-2e reference points shall be integrity protected.

- [CAPIF-SEC-4.4-c] The transport of messages over the CAPIF-2 and CAPIF-2e reference points shall be protected from replay attacks.

- [CAPIF-SEC-4.4-d] The transport of messages over the CAPIF-2 and CAPIF-2e reference points shall be confidentiality protected.

- [CAPIF-SEC-4.4-e] Privacy of the 3GPP user over the CAPIF-2 and CAPIF-2e reference points shall be protected.

- [CAPIF-SEC-4.4-f] The API exposing function shall determine whether API invoker is authorized to access service API.

## 5.2 Key Issue #2: Checking authorization before allowing access

### 5.2.1 Key issue details

Resource owners need to be able to control access to their resources. In the use cases described in TR23.700-95, the resource owner is the UE's user, or the UE's user has been given permission by the subscriber to authorize access to the resource.

### 5.2.3 Potential security requirements

1. Authn-1-ResOwner: when giving or revoking authorization, the resource owner shall be authenticated.
2. Authz-1-General: Access to resources of the resource owner via the northbound APIs shall only be allowed if the resource owner has authorized it.
3. Authz-2-App: Authorization shall be given to an application. Authentication of applications by the operating system of the UE is out of 3GPP scope.

Authz-3-OtherSub: In case it is not the resource owner triggering the AF to invoke an API, the triggerer UE of the AF shall be authorized by the resource owner to access the resource through the API.

Editor's Note: this requirement is FFS taking into consideration the reply from SA6.

1. Authz-4-Scope: The 5G system shall be able to limit the scope of API requests to resources owned by a resource owner.
2. Authz-5-Revoke: The resource owner shall be able to revoke authorization at any time. From then on access to resources based on the revoked authorization shall not be allowed.
3. The 5G system shall be able to preserve the confidentiality of the UE's external identity (i.e., MSISDN) against a third party.

## 5.X Key issue #X: <Title>

### 5.X.1 Key issue details

### 5.X.2 Threats

### 5.X.3 Potential security requirements

# 6 Proposed solutions

## 6.0 Mapping of solutions to key issues

Table 6.0-1: Mapping of solutions to key issues

|  |  |  |  |
| --- | --- | --- | --- |
| Solutions | KI#1 | KI#2 |  |
| Solution #3: UE Originated API invocation using OAuth Client Credential Grant | x |  |  |
| **Solution#6:** Authorization before allowing access to resources |  | x |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 6.1 Solution #1: Resource Owner Authorization in API Invocation using OAuth Token

### 6.1.1 Introduction

This solution addresses the requirement in KI#2.

This solution proposes to reuse OAuth 2.0 with authorization code grant model to obtain resource owner’s authorization in case that an API invocation of network exposure is to process the resource owner’s data.

NOTE1: how the API invoker is authenticated to the authorization server is out of scope of this solution.

NOTE2: how the Resource owner is authenticated to the authorization server is out of scope of this solution.

### 6.1.2 Solution details

### 6.1.2.1 Architecture



Figure 6.1.2.1-1 architecture for CAPIF with SNA enhancement

The architecture is derived from solution 2 in TR 23.700-95 [3]. Definition of the authorization function, CAPIF-8, CAPIF-10, and CAPIF-10e is the same. Triggerer has the same definition of resource owner client(s).

It is proposed that the authorization function is collocated with the CCF.

However, the difference is that CAPIF-9 is not needed in this solution because there is no communication with AEF. CAPIF-8 is application layer, which is out of 3GPP

### 6.1.2.2 Procedure



Figure 6.1.2.2-1 Procedure of Obtaining Resource owner Authorization

As shown in the Figure 6.1.2.2-1, the details of obtaining resource owner authorization in API invocation is summarized as following:

1. API invoker obtains authentication and authorization method (e.g. method 1: TLS-PSK, or method 2: PKI, or method 3: TLS with OAuth token) as specified in clause 6.1 in TS 33.122 [5].

NOTE 3: CCF needs to pre-configure with TLS with OAuth token method.

NOTE 4: Onboarding procedure is reused.

2. API invoker discovers service API as specified in clause 6.3.1.3 in TS 33.122 [5].

3. Resource owner triggers the API invocation. If resource owner authorization is needed for the invoked API and the method 3: TLS with OAuth token is selected, then the API invoker obtains tokenCAPIF via OAuth 2.0 with authorization code grant model as depicted in clause 6.1.2.3.

4. The API invoker invokes northbound API to the AEF as depicted in clause 6.5.2.1 or 6.5.2.2 in TS 33.122 [5]. The API Invocation message includes tokenCAPIF.

The API invoker is pre-configured with a certificate and use TLS to authenticate with AEF.

5. The AEF verifies the tokenCAPIF in the message, and the profile of this token is depicted in clause 6.1.2.4. The AEF verifies the integrity of tokenCAPIF and shall check whether IDs in the scope are align with the IDs in the API invocation message. If the verification is successful, it means the CCF has authorized the API Invoker to access the API and the resource owner has authorized the API invoker to access its resource.

Editor’s Note: Details of content and verification of token is ffs.

### 6.1.2.3 OAuth 2.0 role mapping

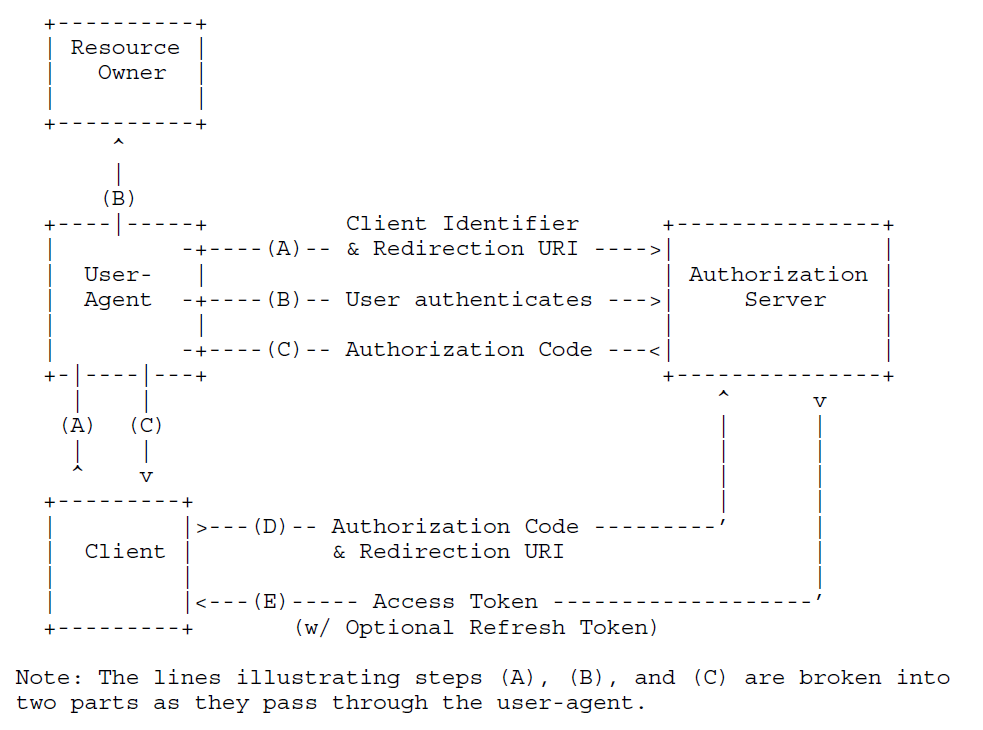


Figure 6.1.2.3-1 Procedure of Obtaining Resource owner Authorization (from RFC 6749 [4])

OAuth 2.0 with authorization code grant model is depicted in clasue 4.1 in RFC 6749 [4]. In this solution, the API invoker endorses the role of client, the triggerer endorses the role of user-agent. The Authorization Function authenticates the resource owner in step B, which depends on existing mechanism and is out of this solution. The API invoker (i.e. client) retrieves tokenCAPIF from the Authorization function in CCF. The scope of the tokenCAPIF includes API invoker ID, service API ID and resource owner ID (i.e. GPSI).

Authorization can be revoked according to mechanism defined in IETF RFC 7009 [7].

The resource owner ID is equal to the UE ID in the API invocation message, e.g. GPSI.

It takes use case 1 defined in Annex A.1 in TR 23.700-95 [3] as an example. An end user (i.e. **resource owner**) is playing a time-sensitive game using a game client application (i.e. **triggerer**) on the end user’s UE communicating with a game server (i.e. **API invoker**), and wants to have a high-quality and low-latency communication for better service experience.

In step A, the end user requests the game client application, and the game client application requests game server to try to invoke the QoS API via **application layer**. The game server discoveries QoS API, and initiates OAuth procedure by contacting the game client application, and the game client application sends the requests to authorization server via **CAPIF-8**.

In step B, the Authorization Server requests the game client application to do user authentication and authorization via **CAIPF-8**, the end user may type in his MNO username and password and click “consent” for extra charging for QoS enhancement.

NOTE: It is assumed that authorization server has linkage between resource owner and GPSI.

Editor’s Note: How Authorization Function maps username to ID of the UE that the user is using when the user has multiple subscriptions is FFS.

In step C, after successful authentication and authorization, the authorization server provides authorization code to the game client application via **CAPIF-8**, and the game client application sends the authorization code to the game server via **application layer**.

In step D, the game server sends authorization code to authorization server via **CAPIF-10/CAPIF-10e**.

In step E, the game server gets tokenCAPIF from authorization server. The game server uses the tokenCAPIF to invoke QoS API via **CAPIF-2/CAPIF-2e** to modify the end user’s QoS.

Editor’s Note: More clarification of Integration of out scope and in scope messages is FFS.

6.1.2.4 TokenCAPIF Profile

The tokenCAPIF is protected by the JSON signature profile as specified in IETF RFC 7515 [8].

The claim in the tokenCAPIF includes the parameters defined in C.2.2 in TS 33.122 [5]. In addition, “resource owner ID” related to the list services is also added in scope. The resource owner ID is equal to the UE ID in the API invocation message, e.g. GPSI.

### 6.1.3 Evaluation

The solution works when the TLS with OAuth token is selected.

This solution assumes that the resource owner is human user.

This solution addresses the requirements Authz-1, Authz-2, Authz-3 and Authz-4, but does not address the following requirements in KI#2:

This solution doesn’t touch authentication of the resource owner or API invoker by the authorization server, it may be addressed by other solution.

This solution doesn’t touch authentication between API invoker and CCF and authentication between API invoker and AEF, it may be addressed by other solution.

This solution doesn’t touch revocation of authorization, it may be addressed by other solution.

This solution doesn’t touch privacy of the UE's external identity against the third-party, it may be addressed by other solution. This solution doesn’t address a static token claim issue after token revocation.

Editor’s Note: Mapping to SA6 defined use case is ffs.

Editor’s Note: Further evaluation is FFS

## 6.2 Solution #2: Authentication using OpenID Connect

### 6.2.1 Introduction

Authentication of the UE can be performed by including an OpenID Connect identity token. In OpenID Connect, the authorization server is usually also the OpenID provider. This solution separates these roles into the authorization function (AZF) and the authentication function (ANF).

### 6.2.2 Solution details



Figure 6.2.2-1: OpenID Connect for SNAAPP AFs

The flow follows the authorization code flow of OpenID Connect [6]. All communication is over TLS secured connections.

1. Prior to any interaction, AZF should be registered with ANF, giving information that will be provided to the UE when requesting authentication.

2. The UE accesses the AZF without identity token

3. The AZF shall redirect the UE to the ANF\_URI, with the type set to code (to indicate authorization code flow), scope set to openid (to indicate that this is an openID request), client\_ID set to its own ID, redirect\_URI set to the URL to be used in step 8, and may set state to some state that can be used by AZF to reduce its internal state. Because sometimes it is necessary to authenticate the subscriber, and sometimes authentication of user is sufficient, the ANF shall offer separate endpoints for user and for subscriber authentication. The AZF shall be aware of which entity shall be authenticated based on configuration. Which authentication method is then chosen is between negotiated between UE/user and ANF and is out of scope of this solution.

4. The UE shall access the ANF at the redirected to ANF\_URI, including the parameters of step 3.

5. The ANF shall authenticate the UE. This can be done in the usual way the home network authenticates the user, which may be GBA, AKMA, or using a proprietary method such as username/password.

6. The ANF should request authorization to release the required personal information to the AZF from the user. In this setup, the ANF presents to the user the information that was made available to the ANF about the AZF in the enrolment.

7. The ANF shall redirect the UE back to the AZF at the AZF\_URI, including as code a single use token, which shall be specific to the AF, and as state the state that was set in step 3.

8.-9. The AZF shall request the identity token from the ANF at its token endpoint. The ANF shall verify that the identity token is indeed being requested by the correct AZF, and, if correct, return the identity token to the AZF.

10. The AZF shall verify the validity of the identity token according to RFC 6749, especially Sections 4.1.2 and 10.12.

Editor's Note: which identifiers are to be used is FFS.

### 6.2.3 Evaluation

1. This solution can be used to address Authn-1-ResOwner.

Editor's Note: further evaluation is FFS

## 6.3 Solution #3: UE Originated API invocation using OAuth Client Credential Grant

### 6.3.1 Introduction

According to KI#1 and the SA1 requirement referenced in KI#1, a UE shall be able to access a northbound API of the 5G system. The API invocation is triggered by an application, which is not visible to the 5G system.

Therefore, it is assumed that details of the interface between the application and the API invoker on the UE are out of scope of this solution.

It is proposed to use the OAuth client credential grant (specified in clause 4.4. of [4]) as a basis for the solution.

### 6.3.2 Solution details

The context of the components of the solution is visualized in Figure 6.3.2-1.

It is assumed that the application is deployed on the mobile device consisting of the UE. For instance, the application might be an app installed on a smartphone. Note, whether the application is considered being part of the UE or being collocated with the UE on a mobile device is a matter of interpretation and not relevant for the solution.

The API Invoker on the UE is taking the role of the OAuth Client. If CAPIF is applied, the Authorization Server is part the CAPIF Core Function. In this case the interface between API Invoker and Authorization Function is part of the CAPIF-1 interface.

CAPIF onboarding for assignment of Client Id and Client secret, since Client Id and Client secret can be derived from the 5G key hierarchy as explained in more detail in step 2 of the message flow below.



Figure 6.3.2-1: Solution components of UE originated API invocation using OAuth client credential grant.

A typical message flow executed for UE originated API invocation is depicted in Figure 6.3.2-2.



Figure 6.3.2-2: Message flow used for UE originated API invocation using OAuth client credential grant

The individual steps are described below:

1. The flow is triggered by the application sending a request to the API invoker on the UE.

The interface between Application and API invoker including the authorization of the Application to use this interface is out of the scope of this solution.

It is assumed that this interface can be handled like other internal APIs on a mobile device, like access to cameras or GPS location.

2. Based on the received Request the API invoker on the UE (acting as an OAuth Client) sends an OAuth Token Request to the Authorization Server with the grant type set to "client\_credentials".

The Token Request is sent as https request. As part of the session establishment the API invoker authenticates the Authorization Server by verifying the Authorization Server's server certificate.

The FQDN of the authorization server can be constructed in a way standardized by 3GPP. The operator can obtain a server certificate from one of the CAs, which are trusted by the operating system of the UE and whoose root certificate are preinstalled in the operating system of the UE.

The scope parameter is set by the API invoker based on the request of the application such that it covers the necessary scope of the subsequent API request in step 5.

In line with the SA1 requirement that the Application itself is invisible to the 5G system the token request does not contain information related to the identity of the Application, but the solution introduces a new entity, API invoker, in the UE, which is visible to the 5G system.

The Token Request includes the authentication of the API Invoker using the API invoker's client credentials. The circumstance that the API invoker is located and associated with a UE can be exploited for instance by using AKMA

In this context the Authorization Server takes the role of the AKMA AF. The https protocol is used as Ua\* protocol. Depending on whether the Authorization Server is regarded as trusted the Authorization Server can directly interact with the AAnF or via NEF. Depending on the choice the SUPI or the GPSI can be used as OAuth client Id. The AKMA key KAF can be used as client credential. Thus, the authorization header of the http Token Request sent from the UE to the authorization server can be used for http Basic authentication with A-KID and KAF as username and password, respectively. Alternatively, also http Digest authentication can be used. This can be decided during normative work.

3. The authorization server is authorizing the request based on the identity of the API Invoker and the requested scope by applying stored policies and permissions.

Note, this solution assumes that policies and permissions are pre-arranged at the authorization server. Since the application is not visible, the policies and permissions are in the granularity of UE level, not the application level.

4. In case of successful authorization the authorization server returns the token to the API invoker. The token includes claims, which reflect the granted scopes and permissions.

Editor's Note: Whether 3GPP needs to define additional scopes and claims for token requests and tokens is FFS.

5. The API invoker is sending the actual API request to the AEF. The API request contains the token received in the previous step.

The API Request is sent as https request. As part of the session establishment the API invoker authenticates the AEF by verifying the Authorization Server's server certificate.

If mutual authentication is needed, AKMA can be used between API invoker and AEF (e.g. https digest authentication).

6. The AEF is validating the incoming request. That is, the AEF verifies the validity of the token, and the AEF verifies that the request is within the scope described by the claims in the token.

Editor's Note: It is FFS, if the AEF can validate the scope of the API request solely on the content of the token or if other mechanisms are needed and, if so, if further standardization is required for this purpose.

7. In case of successful validation the AEF is executing the request by involving other NFs (not shown in the signalling diagram)

8. The AEF returns the result of the API call to the API invoker.

9. The API invoker completes the flow by sending a response to the application (which is based on and might include results received in the previous step.

NOTE: Authorization from the resource owner (i.e., the user) is not considered as part of the OAuth procedure using Client Credential Grant type.

### 6.3.3 Evaluation

1. The presented approach provides a solution for the case that API Invoker is part of the UE and that this API Invoker can be used by applications on a mobile device to utilize 5G northbound APIs. The approach is thus complementary to other solutions which target the case that the API Invoker is part of the third-party application.
2. Usage of AKMA guarantees that the client Id of the API Invoker is really bound to the Id of the UE.
3. In ths solution the 5G system only authenticates and authorizes the UE. The 5G system does not authenticate or authorize the (invisible) third party application or a user. That is, the solution assumes that the UE is the resource owner or requesting party.
4. In case of one UE accessing resources of another UE, the subscriber of the second UE is not able to authorize the third-party application used on the first UE. Such scenarios can be avoided, if API calls are restricted to the scope of each UE and API invocation crossing UEs is handled on the application layer.
5. The main changes required for the solution are related to the deployment of the API Invoker on the UE and the definition of interface between API Invoker and the third-party application. However, this interface is out of scope of this solution and out of scope of SA3.
6. Potential changes might also be needed with respect to definition of additional scope in the northbound APIs and the way how an AEF is able to restrict the scope of API calls to the invoking UE.

Editor's Note: Furhter evaluations are FFS.

## 6.4 Solution #4: Authenticate and authorize UE in UE originated API invocation

### 6.4.1 Introduction

The solution addresses key issue #1. For originated API Invocation, the solution describes how UE (an API Invoker) can be provided with secure access to APIs (e.g., triggered by an application that is not visible to the 5G system), by authenticating and authorizing the UE.

### 6.4.2 Solution details



Figure 6.4.2-1: UE originated API Invoker authentication, authorization, and secure connection establishment process

The steps shown in Figure 6.3.2-1 is described as follows:

Precondition: The UE (i.e., an API invoker) can be registered to the network.

Steps 1-6 API Invoker Onboarding:

The UE is provisioned with a CAPIF Core Function (CCF) information such as CCF address/ID (e.g., after a successful primary authentication in any protected message). The UE and the network can derive the onboarding enrolment information such as CAPIF security credentials i.e., a CCF key (based on UE 5G security context e.g., AKMA key/AUSF key). The UE can derive a CCF key (Kccf) and key identifier (Kccf ID) which can be used to authenticate and establish a secure communication (e.g., TLS PSK based on CCF Key) with the CCF during the onboarding process.

NOTE 1: The 5G security key used to generate CAPIF security key and the input used are upto the normative work.

1. The API Invoker can send an Onboarding Service request to the CCF which can include Onboarding type (i.e., Subscriber Indication or UE service based), Kccf ID, UE ID (e.g., GPSI).

2. Based on the received Onboarding type, the CCF determines to fetch security context related to the UE. The CCF can send a key request to the Core NF/AF which can include UE ID, and Kccf ID

Further the Core NF/AF provides the SUPI, and Kccf to the CCF in a Key Response message.

3. The CCF stores the CCF key, Kccf ID and UE ID.

4. The API Invoker and the CCF can perform TLS authentication and establish secure session based on Kccf shared between API Invoker and CCF.

5. With a secure session established, the API Invoker sends an Onboard API Invoker Request message to the CCF which includes the UE ID (SUPI/GPSI).

6. The CCF generates an API invoker's profile and onboard secret as specified in TS 23.222. The AEF Key can be used by the API Invoker to authenticate and establish secure session with the AEF as in TS 33.122 Clause 6.5.2.3. The AEF key can be derived from Kccf and other input parameters: API Invoker ID, CCF ID, Target AEF ID(s)/information, and Nonce.

The CCF on a successful authentication and authorization, it can locally store the API Invoker profile with API Invoker ID, Onboard Secret, AEF key along with Target AEF information.

The CCF can respond with an Onboard API invoker response message same as TS 33.122 which can include the CAPIF core function assigned API invoker ID, AEF Authentication and authorization information, and/or Onboard Secret, along with other information such as nonce, AEF information (Target AEF ID.

The API Invoker stores information received from step 6 and the API invoker is onboarded.

API Invoker can perform CAPIF-1 authentication anytime with CCF based on TS 33.122.

Steps 7-9 Access Token Request/Response:

7. The API Invoker sends Oauth 2.0 based access token request as in TS 33.122 Clause 6.5.2.3 (i.e., with grant\_type client credentials if the API Invoker is the Resource Owner.

If the API Invoker is not a resource owner, for OAuth 2.0 access token request the grant\_type may be Authorization Code Grant as described in RFC 6749.

Similar to TS 33.122 Clause 6.5.2.3, the API invoker may include the CAPIF core function assigned API invoker ID and the Onboard\_Secret in the OAuth access token request message for the CAPIF core function to validate the access token request.

8. The CCF based on the local policy checks if there exists any related authorization or prior consent information managed in the network related to allowing the API invoker to consume any service API invocation related to the UE.

NOTE 2: The collection and management of user consent or authorization information related to a service exposure is outside the scope of this solution. It is assumed that the network manages such authorization information in any storage function outside the scope of this solution.

9. The CCF generates and sends an Oauth access token (based on OAuth 2.0) to be used as AEF Access token. The access token claims can include UE ID and CCF ID along with other claims in TS 33.122 Clause C.2.2.

Steps 10-15 Service API Invocation:

10. The API Invoker derives an AEF Key (Kaef) from CCF Key (Kccf) and respective input parameters similar to CCF on a successful CAPIF 1 authentication and authorization. The API Invoker can send Authentication Initiation Request to the AEF, which includes the CCF assigned API invoker ID, and UE ID.

11. The AEF can send API Invoker ID, and UE ID to request the security information from the CCF. The CCF provides the security information related to the chosen security method (e.g., TLS-PSK: AEFPSK) along with AEF Key, Service API(s) authorization information (can be a list of Service APIs which can be invoked by the API Invoker related to the UE ID), and Oauth access token i.e., AEF Access token (to authorize the API invoker to request the service API invocation from AEF) to the AEF over CAPIF-3 reference point. The CCF can provide the remaining validity timer value for the AEF Key (i.e., AEFPSK) as in TS 33.122 Clause 6.5.2.1.

Alternatively, the service information and access token sending can be skipped as it can be bound to the access token as claims (later received in step 13a) from the API invoker.

12. After fetching the relevant AEF Key for the authentication, the AEF can send Authentication Initiation Response message to API invoker to initiate the TLS session establishment. The AEF starts the validity timer based on the value received from the CAPIF core function in step 11.

The API Invoker and the AEF can perform mutual authentication using the AEF key and establish TLS session.

13a. The API invoker can send Invocation service request to the AEF which can include Requested Service API(s) information, API Invoker ID, UE ID, and AEF Access Token (received from CCF).

13b. The AEF can authorize the API invoker's service API invocation request based on authorization information (i.e., validating the claims in the Oauth based AEF Access Token) obtained from CAPIF core function as specified in subclause 8.16 of TS 23.222.

14. On a successful access token valdiation, the AEF considers Invocation service request authorization as successful, execute API request and can send Invocation service response with success indication.

NOTE 3: According to TS 33.122, API provider domain provides Onboarding enrollment information to the API invoker as a prerequisite to the Onboarding procedure. Therefore, for the UE originated API Invocation case, the solution enables provision of onboarding enrollment information based on UE’s established security context as described in steps 1.

NOTE 4: The user consent information collection and related management is outside the scope of this solution.

### 6.4.3 Evaluation

The solution addresses KI#1 and enables the following for the case where the API Invoker resides in a UE:

A CCF key need to be derived and provided to the CCF by a NF or an AF.

API Invoker UE and CAPIF Core Funtion (i.e., Authorization Function)mutual authentication based on a key derived from UE 5G security context.

TLS-PSK based mutual authentication and secure connection establishment between API Invoker UE and the AEF is enabled.

OAuth 2.0 token based authorization is used for access to service API.

Editor’s Note: Further evaluation is FFS.

## 6.5 Solution #5: Resource Owner based authorization for resource access

### 6.5.1 Introduction

The solution addresses Key Issue #2.

The solution describes the method to receive and revoke authorization as required from an authenticated Resource owner to control access to resource(s) of a resource owner. Figure 6.5.2-1 shows the resource authorization procedure to allow access to resources.

### 6.5.2 Solution details

Figure 6.5.2-1: Resource authorization procedure to allow access to resources

The steps shown in Figure 6.5.2-1 is described as follows:

1-2. During the primary authentication procedure, the UDM may also indicate to the AUSF whether CAPIF keys need to be generated for the UE (i.e, resource owner). If the CAPIF Indication is included, the UDM may also include GPSI and CAPIF Function information (i.e., ID/address).

3. If the AUSF receives the CAPIF indication from the UDM, the AUSF and UE following a successful primary authentication can generate CAPIF Key and an ID (to identify the CAPIF Key for the UE) from the AKMA Key.

NOTE: The inputs used in the CAPIF key and ID generation can be upto the normative phase.

4a. The AUSF based on CAPIF Function information, sends resource owner authorization data notification request to the CAPIF function which can include the UE ID (i.e., GPSI), CAPIF key and the ID.

4b. The CAPIF Function stores the received information and sends a sends resource owner authorization data notification response to the AUSF.

5. The AUSF sends the CAPIF Function information and GPSI to the AMF and the AMF forwards the information to the UE over the NAS transport.

6a. The UE initiates a resource owner data notification trigger with GPSI and a freshness parameter to the CAPIF function.

Optionally, prior to step 6a, the UE may trigger resource owner registration request with the CAPIF function by sending a GPSI and then can establish a secure connection based on CAPIF Key. The CAPIF function then sends a resource owner ID and lifetime (for the resource owner registration) following a successful resource owner registration.

6b. The CAPIF function and the UE (resource owner) establishes secure connection using the resource owner key generated from the CAPIF Key, the freshness parameter and using and other inputs if requried.

7. Resource owner authorization data can be fetched or handled over the established secure connection. The resource owner authorization data fetching, and handling can be upto operator implementation.

### 6.5.3 Evaluation

The solution addresses Key Issue #2 and enables the following:

An application key derived from CAPIF Key related to the Resource Owner is used to establish secure application session with the CAPIF Function.Access to resource of resource owner is allowed based on resource owner specific authorization information (i.e., user consent data) verification by the network.

The solution assumes the resource owner as the subscriber, therefore the UE is considered as resource owner UE, and the client/application in the UE is considered as a resource owner’s client/application.Editor’s Note: Mapping to SA6 defined use case is ffs.

Editor’s Note: Further evaluation is FFS.

## 6.6 Solution #6: Authorization before allowing access to resources

### 6.6.1 Introduction

This solution addresses the security requirement about authorization by the resource owner before allowing access to resources of the resource owner, which is detailed in key issue #2.

It is assumed that authorization information by the resource owner has been received and stored in the authorization server with a method out of the scope of this solution.

Also, the consideration whether the resource owner is the subscription user or the subscription owner is out of scope of this solution.

The MNO learns the authorization information from the subscription user or from the subscription owner and stores the authorization information, which is bound to the UE identifier, in the PLMN trusted domain. How the MNO authenticates the resource owner and learns the authorization information is out of scope of this solution.

This solution covers the case that the API invoker is the AF accessing to resources related to a UE or the API invoker is the application in the UE accessing to resources related to that UE.

How the AF maps the target username in the application layer into the UE identifier is out of scope of this solution. The authentication and authorization behind the AF-CAPIF interaction for the triggering UE and user is out of scope.

This solution assumes that the authorization server is co-located with the CAPIF Core Function (CCF). This solution does not specify the place where the authorization information is stored. The CCF may store the authorization information in an external storage, and in this case it is assumed that there is a secure channel between the CCF and the external storage.

Editor’s Note: Clarification of storage is FFS.

Editor’s Note: Consideration of user cases if FFS.

### 6.6.2 Solution details

Below describes the steps of the procedure for "authorization before allowing access to resources", which is shown in Figure 6.6.2-1.



Figure 6.6.2-1: Authorization before allowing access to resources

How the authentication is executed for the API invoker that runs in the UE is out of scope of this solution. In general, the solution doesn’t focus on the authentication of the API invoker.

1. The API invoker and the CCF execute authentication procedures and establish a secure channel as specified in TS 33.122 [5].
2. The API invoker request OAuth access token as specified in TS 33.122 [5].
3. The CCF verifies the request.

Steps 4-6 is executed if resource owner authorization check is needed for the API invocation.

1. The CCF may fetch resource owner authorization information from the storage.
2. The storage sends the resource owner authorization information.
3. The CCF issues an access token that includes an indication for the resource owner authorization. The CCF sends the issued token to the API invoker.
4. The API invoker and API exposing function establish a secure channel.
5. The API invoker sends the token to the API Exposing Function.
6. The API Exposing Function verifies the token and checks the resource owner authorization information before allowing access to the resources related to the subscription.
7. The API Exposing Function returns the API invocation response to the API invoker.

For the UE originated API invocation case where an application triggers the API invocation by the application in the operating system in the UE, this solution assumes that an authorization in the granularity of application level is executed by a mechanism, like allowing users in the mobile phones to control the permission of application to access resources such as microphone of the mobile phone, provided by the operating system, which is out of scope of this solution. To give the access control power to the MNO considering the permission from the user or subscriber, the MNO needs to retrieve the permission/authorization information from the subscriber or user and store it in the authorization server/storage with an out of scope mechanism. This solution applies to the specific case where the application is accessing to the resources of the UE on which the application is running. The case of accessing resource of other UEs by the UE is not covered in this solution.The following procedure, depicted in Figure 6.6.2-2, shows how the case that the resource owner revokes the authorization after the CCF issues a token can be handled.



Figure 6.6.2-1: Revocation of resource owner authorization

1. Resource owner authorization is revoked.
2. The CCF informs the AEF about the revocation. (It is assumed that the AEF has subscribed to the CCF event exposure service).
3. The API invoker and AEF executes some authentication and establish a secure channel using TLS.
4. The API invoker sends the access token in the NB API call.
5. The AEF verifies the access token, checks the resource owner authorization. Since the resource owner authorization has been revoked, the AEF rejects the request.
6. The AEF sends the rejection response to the API invoker.

### 6.6.3 Evaluation

The solution addresses the following cases:

* The AF accesses resources related to a UE
* The application in the UE is accessing the resources related to that UE.

The solution assumes that there is a mechanism in the UE for authorization in the application-level granularity.

This solution is a future proof solution considering possible extension of the definition of resource owner to cover the subscribers in addition to the users.

This solution assumes that the API invoker application in the operating system of the UE is authenticated and authorized by a method out of scope.

Editor’s Note: Further evaluation is FFS.

## 6.7 Solution #7: Authorizing UE originated API invocation with PKCE flow

### 6.7.1 Introduction

In case the API invocation can be initiated by an application on the UE without making use of a unique CAPIF client agent per UE, the UE application might not be able to securely store a client credential. For this case, there is the Authorization Code Flow with Proof Key for Code Exchange defined in RFC 7636 [9].

### 6.7.2 Solution details

The solution uses the PKCE protocol flow with the following mapping: the client in RFC 7636 is the application on the UE. The authorization server in RFC 7636 is the authorization function in the network.

The following figure gives an example PKCE flow to help understanding the concept of the PKCE flow. This flow could look different for a different authentication mechanism.



Figure 6.7.2-1: example PKCE flow

Editor's note: whether a separate onboarding process is necessary for UE originated API invocation is FFS.

### 6.7.3 Evaluation

Editor's note: whether the user interaction required for PKCE flow is acceptable for SA6 is FFS.

## 6.8 Solution #8: Validation of OAuth Token

### 6.8.1 Introduction

If an OAuth token used for authorization, the API exposing function needs to verify the validity of the token. In this solution, this is done by token introspection RFC 7662 [10]. The revocation procedure between authorization server and resource owner is out of scope of this solution.

### 6.8.2 Solution details



Figure 6.8.2-1: validation of OAuth 2.0 token

All messages containing a tokens shall be protected using TLS. The API invoker shall authenticate the API exposing function by verifying the API exposing function's certificate. The API exposing function shall authenticate the authorization function by validating the authorization function's certificate

Editor's Note: contents of the certificates and which CAs are acceptable is FFS.

The oAuth access token shall contain:

- a unique random string,

- which API the token applies to (the scope),

- who is the resource owner,

- the API invoker ID, and

- expiry time.

Editor's Note: how that information is encoded in the OAuth access token is for stage 3.

1. The API call shall contain the OAuth access token.

2. The API exposing function shall verify that the access token is applicable to the desired API call and if yes, send the access token to the authorization function for validation. Otherwise the API call shall fail and a new authorization may be requested.

3. The authorization function shall verify the validity of the access token and return whether the token is valid.

4. If the access token is valid, the API exposing function shall execute the API call.

The API exposing function may cache the result of validation. In that case, the API exposing function shall subscribe to receive a notification in case the token is revoked. This subscription may be included in step 2, e.g. by accessing a different endpoint for validate and subscribe than for validate only.

The authorization function shall offer a notification service to inform the API exposing function of revokation of a token. Subscription shall be on a per token basis.

The authorization function may store information about validity of tokens locally.

### Editor's Note: whether caching is required is FFS.6.8.3 Evaluation

This solution addresses Authz-4-Scope and Authz-5-Revoke. This solution deviates from the existing CAPIF solution in clause 6.5.2.3 in TS33.122 [5]. It can avoid the impact of signature generation and verification and of relying on synchronized time and short lived tokens at the expense of backend communication between AEF and authorization function.

Editor's Note: whether token introspection provides benefits over signature verification is FFS.

## 6.9 Solution #9: OAuth 2.0 based API invocation procedure

### 6.9.1 Introduction

This solution addresses the key issue #2 in terms of API invoker UE authorization.

* In this solution, API invoker may request authorization for service API and resource. And only one access token is provided to the API invoker for service API and resource authorization.Resource owner can authorize API invoker for resources via the UE. The resource owner can authorize the API invoker to access its resources in synchronous and asynchronous manner.
* CAPIF core function can authorize API invoker for services and service operations.

### 6.9.2 Solution details



Figure 6.9.2-1: OAuth 2.0 based API invocation.1.API invoker and UE controlled by the resource owner do the mutual authentication. Resouce owner agnet is a part of the UE.

For the case that API invoker is the UE, the mutual authentication can be realized based on certificates. For the case that API invoker is the AF, the mutual authentication can be realized based on GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism. A secure connection between API invoker and resource owner is established after the mutual authentication. The secure connection can be established via TLS.

2. The API invoker sends authorization request to the UE controlled by the resource owner.

In case API invoker has obtain the authorization of service API and service operation, the request includes the identity (e.g., GPSI, IMPI or application layer ID) of the API invoker, the identity (e.g., GPSI, IMPI or application layer ID) of target UE, the target resource (e.g., location of UE, QoS of the UE).

In case API invoker has not obtain the authorization of service API and service operation, the request includes the identity (e.g., GPSI, IMPI or application layer ID) of the API invoker, the identity (e.g., GPSI, IMPI or application layer ID) of target UE, the target resource (e.g., location of UE, QoS of the UE), service identifier, service identifier (e.g., Nnef\_ParameterProvision), service operation identifier (e.g., Nnef\_ParameterProvision\_Update). The service identifier/service operation identifier indicates the service/service operation that can be performed on the target resource.

3. CAPIF core function/authorization function and UE controlled by the resource owner should do the mutual authentication.

For the case of CAPIF core function, resource owner can authenticate CAPIF core function via certificate. Then CAPIF core function can authenticate resource owner using GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism. And CAPIF core function may generate certificate and OAuth 2.0 token for the resource owner after the authentication.

For the case of authorization function, resource owner can authenticate authorization function via certificate. Then authorization function can authenticate resource owner using TLS-PSK, OAuth token, GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism, in which the certificate can be assigned by the CAPIF core function.

A secure connection between resource owner and CAPIF core function/authorization function is established after the mutual authentication. The secure connection can be established via TLS.

CAPIF core function and resource owner should also do the mutual authentication. Detials on the authentication procedure is out of scope of 3GPP.

4. The resource owner may timely and synchronously grants the authorization request for the resource via the UE. And the UE controlled by the resource owner sends the authorization request and grant information to the CAPIF core function/authorization function to request the authorization code.

The resource owner may asynchronously grant the authorization request for the resource based on the local pre-generated profile. And the resource owner sends the authorization request and grant information to the CAPIF core function/authorization function to request the authorization code.

If the resource owner previously sends the pre-generated profile to the CAPIF core function/authorization function, the resource owner sends the authorization request to the CAPIF core function/authorization function to request the authorization code. CAPIF core function/authorization function can authorize the authorization request for the resource based on the pre-generated profile.

The authorization request in step 3 is identical to the one sent by the API invoker.

If the API invoker has obtained the authorization of service API and service operation, the CAPIF core function/authorization function generates the authorization code for the API invoker when API invoker is authorized to request the resource.

If the API invoker has not obtained the authorization of service API and service operation, the CAPIF core function/authorization function should check if API invoker is authorized to invoke the service API and service operation based on pre-configured policies. If the API invoker is authorized to invoke the service API, service operation, and the resource, the CAPIF core function/authorization function generates authorization code for the API.

5. CAPIF core function/authorization server sends the authorization code to the UE controlled by the resource owner.

6. The UE controlled by the resource owner sends the authorization code to the API invoker.

7. CAPIF core function/authorization function and API invoker should do the mutual authentication.

For the case of CAPIF core function, API invoker can authenticate CAPIF core function via certificate. Then CAPIF core function can authenticate API invoker using GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism. And CAPIF core function may generate certificate and OAuth 2.0 token for the API invoker after the authentication.

For the case of authorization function, API invoker can authenticate authorization function via certificate. Then authorization function can authenticate API invoker using TLS-PSK, GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism, in which the certificate can be assigned by the CAPIF core function.

A secure connection between API invoker and CAPIF core function/authorization function is established after the mutual authentication. The secure connection can be established via TLS.

8. The API invoker sends the authorization code to the CAPIF core function/authorization function.

9. The CAPIF core function/authorization function sends the refresh token/access token to the API invoker. The API invoker can send the refresh token to CAPIF core function/authorization function to obtain access token.

The access token includes CAPIF core function identity (e.g., NF instance ID, NF ID), authorization function identity (e.g., NF instance ID, NF ID), AEF identity (e.g., NF instance ID, NF ID), service API identifier (optional), service identifier(optional), API invoker identity (e.g., GPSI, IMSI, application layer ID), resource owner identity (e.g., GPSI, IMSI), user resource identifier (e.g., location), expire time.

10. API Exposing function (AEF) and API invoker should do the mutual authentication.

API invoker and API Exposing function can do mutual authentication based on TLS-PSK, OAuth token, GBA-based authentication mechanism, AKMA-based authentication mechanism, or certificate-based authentication mechanism.

A secure connection between API invoker and API Exposing function is established after the mutual authentication. The secure connection can be established via TLS.

11. The API invoker sends service API invocation request to the AEF. The request includes the API invoker identity, the resource owner identity, the service API that needs to be invoked, the user resource identifier that the API invoker needs to access, and the access token.

12. The AEF authorize the request based on the token.

13. The AEF sends response to the API invoker.

Editor's Note: Reference to the existing procedure for step 1, 2, and 6 are ffs.

### 6.9.3 Evaluation

TBD

## 6.10 Solution #10: UE credential based API invocation procedure

### 6.10.1 Introduction

This solution addresses the key issue #2 in terms of API invoker UE authorization.

In this solution, CAPIF core function/authorization function can authorize API invoker UE to access resources based on the verified UE identity in step 1.

The API invoker UE credential can be certificate or secrets shared among API invoker UE and CAPIF core function/authorization function.

The shared secrets among UE and CAPIF core function can be shared keys related to AKMA and GBA methods.

The shared secrets among UE and authorization can be shared keys related to AKMA method, GBA method, and CAPIF onboarding procedure.

### 6.10.2 Solution details

It is assume that the API invoker is an UE in SNA scenarios.



Figure 6.10.2-1: UE credential based API invocation procedure.

1. The API invoker and authorization function/CAPIF core function do the mutual authentication.

For the case of CAPIF core function, the API invoker can authenticate CAPIF core function via a certificate. Then the CAPIF core function can authenticate the API invoker using GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism. And the CAPIF core function may generate a certificate for the API invoker after the API invoker onboarding procedure.

For the case of authorization function, the API invoker can authenticate the authorization function via a certificate. Authorization function can authenticate the API invoker UE based on its credential (e.g., KAF). In specific, the authorization function can authenticate API invoker using GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism, in which the certificate can be assigned by the CAPIF core function. API invoker and authorization function may do the mutual authentication based on methods that are defined in clause 6.5.2 of TS 33.122[5].

A secure connection between the API invoker and authorization function/CAPIF core function is established after the mutual authentication. The secure connection can be established via TLS.

2. The API invoker sends a token request to the CAPIF core function/authorization function. The request includes the identity (e.g., IMPI, GPSI) of the API invoker, the identity (e.g., IMPI, GPSI,) of the resourcw owner, the target resource (e.g., location of UE, QoS of the UE), and the target service API.

3. The CAPIF core function/authorization function sends the token to the API invoker if the verified identity of UE in step 1 is authorized to request the target resource of the resource owner. The authorization can be realized based on policies provided by the resource owner. The type of token can be refresh token or access token.To access the service API via the API exposing function, the API invoker UE needs to send the refresh token to the CAPIF core function/authorization function to obtain the access token when the CAPIF core function/authorization function sends the refresh token to the API invoker UE. The token shall include the CAPIF core function/authorization function FQDN (issuer), API invoker identity (subject), expected service API (Scope), target resource (Scope), identity of resource owner (Scope), API exposing function FQDN (audience), expiration time (expiration). Details on token, refresh token, and access token are given in RFC 6749 [4].4. The API invoker and API exposing function do the mutual authentication. The API invoker can authenticate the authorization function via certificate. Then the authorization function can authenticate the API invoker using GBA-based authentication mechanism, AKMA-based authentication mechanism or certificate-based authentication mechanism, in which the certificate can be assigned by the CAPIF core function. API invoker and API exposing function may do the mutual authentication based on methods that are defined in clause 6.5.2 of TS 33.122[5].

A secure connection between API invoker and API exposing function is established after the mutual authentication. The secure connection can be established via TLS.

5. The API invoker sends a service API request to the API exposing function along with the access token.

6. The API exposing function needs to reject the request when the API invoker sends the service API invocation request without the token. The API exposing function authorizes the request based on the token. The API exposing function can verify the integrity of the token using the public key of the CAPIF core function. The API exposing function can terminate the procedure if the token is modified. Otherwise, the API exposure can check the service API invocation request against the token. The API exposing function should reject the service API request if the request cannot fulfill the requirements of the token. The procedure goes to step 7 if the request fulfills the requirements of the token.

7. The API exposing function sends the service API invocation response to the API invoker.

### 6.10.3 Evaluation

TBD

## 6.11 Solution #11: Providing and Revoking Resource Owner Authorization using OAuth 2.0 Authorization Code Grant

### 6.11.1 Introduction

This solution addresses Key Issue #2 "Checking authorization before allowing access".

This solution proposes to use OAuth 2.0 Authorization Code Grant as specified in clause 4.1 of RFC 6749 [4], in which the resource owner can be authenticated by the resource owner's 3GPP credentials.

UE, API Invoker, and Authorization Function in this solution performs the role of User-Agent, Client, and Authorization server in RFC 6749 [4], respectively.

### 6.11.2 Solution details

### 6.11.2.1 Architecture



Figure 6.11.2.1-1 architecture for CAPIF with SNA enhancement

This solution uses an architecture proposed in solution #2 of TR 23.700-95 [3]. As defined in TR 23.700-95 [3], the Resource owner client is an application client used by end-user or subscriber of the API provider domain's service provider.

### 6.11.2.2 Procedure

Pre-requisite:

1. During the primary authentication, AUSF receives SNAAPPY Indication from UDM, which indicates that the AUSF and the UE need to generate the following pre-requisite 2.
2. After the primary authentication, UE and AUSF generate SNAAPPY Key Identifier (S-KID) and KSNAAPPY from KAUSF as detailed in 6.11.2.3 and 6.11.2.4 of this document, respectively. After the S-KID and KSNAAPPY are generated, AUSF sends the KSNAAPPY, S-KID, and SUPI to Authorization Function. The Authorization Function stores this information sent by the AUSF. If there were KSNAAPPY and S-KID corresponding to the same SUPI, they are overridden by the new KSNAAPPY and S-KID.



Figure 6.11.2.2-1 Procedure for resource owner authorization based API invocation

1. After API Invoker performs onboarding procedure to CAPIF core function as specified in clause 6.1 of TS 33.122 [5], the API Invoker mutually authenticates with the CAPIF core function as specified in clause 6.3 of TS 33.122 [5].
2. API Invoker performs mutual authentication with API exposing function and gets an authorization to invoke a service API using one of three methods specified in clause 6.5 of TS 33.122 [5].
3. For a service API which needs a resource owner's authorization, the API Invoker shall get an authorization from the resource owner in addition to the authorization that was obtained in step 2.
4. The API Invoker obtains Authorization Code via OAuth 2.0 Authorization Code Grant as specified in RFC 6749 [4]. While the API Invoker redirects the UE to the Authorization Function, the UE sends SNAAPPY indicator, which indicates that the UE supports the resource owner authentication with KSNAAPPY, to the Authorization Function. If the Authorization Function decides to authenticate the resource owner using KSNAAPPY, the Authorization Function sends a challenge to the UE. UE responds with the S-KID and a hash signature which is generated using the S-KID, the challenge, and KSNAAPPY. The Authorization Function verifies the hash signature using KSNAAPPY which the Authorization Function can find based on the S-KID. If the verification is successful and then the resource owner authorizes the API Invoker to invoke the service API, the Authorization Function sends Authorization Code to the API Invoker.

NOTE: Although the Authorization Function is illustrated as a separate entity from CAPIF core function in Figure 6.11.2.2-1, it may be deployed within the CAPIF core function according to the decision in SA3.

NOTE: Other authentication method between the resource owner and the Authorization Function can be performed if the UE does not send SNAAPPY indicator.

Editor's Note: Whether authenticating the resource owner using 3GPP credential is sufficient is FFS.

1. The Authorization Function generates an OAuth 2.0 token, TokenSNAAPPY. The TokenSNAAPPY conveys the S-KID and the generated time of the TokenSNAAPPY, in addition to the token claims speicified in Annex C.2.2 of TS 33.122 [5].
2. The API Invoker requests the TokenSNAAPPY from the Authorization Function by presenting the Authorization Code sent by the Authorization Function in step 4.
3. The Authorization Function sends the TokenSNAAPPY to the API Invoker.
4. The API Invoker performs the service API invocation by presenting the TokenSNAAPPY.
5. If the API Invoker requested a service API that needs resource owner's authorization, API exposing function shall check whether the API Invoker presented a TokenSNAAPPY. If the API Invoker performed the service API invocation without TokenSNAAPPY in step 8, the API exposing function shall reject the request. If the verification of the TokenSNAAPPY is successful, API exposing function identifies the UE using the S-KID which is included in the TokenSNAAPPY, by communicating with the Authorization Function that stored the S-KID and SUPI.
6. API Invoker receives the service API invocation response from the API exposing function.
7. If the resource owner does not want for the API Invoker to invoke the service API, the resource owner can revoke the TokenSNAAPPY at anytime even before the validity time of the TokenSNAAPPY.
8. The UE requests the API Invoker to revoke the TokenSNAAPPY for the service API.
9. The API Invoker requests the Authorization Function to revoke the TokenSNAAPPY for the service API as specified in RFC 7009 [7].
10. If the Authorization Function receives a revocation request for TokenSNAAPPY,
11. The Authorization Function responds to the revocation request. The response includes revocation time and the TokenSNAAPPY with a hash signature which the Authorization Function generates using the TokenSNAAPPY, revocation time, and KSNAAPPY; and
12. The Authorization Function notifies the API exposing function of the revocation of the TokenSNAAPPY, with the revocation time. After the API exposing function receives the revocation notification of the TokenSNAAPPY, the API exposing function shall reject the API invocation from the API Invoker if the API Invoker invokes the service API with TokenSNAAPPY of which the generated time is prior to the revocation time.
13. The UE verifies the hash signature using KSNAAPPY. The UE may inform the resource owner of the revocation result based on the verification.

### 6.11.2.3 S-KID

S-KID is in NAI format as specified in clause 2.2 of IETF RFC 7542 [11], i.e. username@realm. The username part includes SNAAPPY Temporary UE Identifier (S-TID), and the realm part includes Home Network Identifier or Authorization Function Address.

When deriving S-TID from KAUSF, the following parameters shall be used to form the input S to the KDF:

* FC = 0xXX;
* P0 = "S-TID";
* L0 = length of "S-TID";
* P1 = SUPI;
* L1 = length of SUPI.

The input key KEY shall be KAUSF.

NOTE: FC value to be determined during normative phase.

### 6.11.2.4 KSNAAPPY derivation function

When deriving KSNAAPPY from KAUSF, the following parameters shall be used to form the input S to the KDF:

* FC = 0xYY;
* P0 = “Authorization”;
* L0 = length of “Authorization”;

The input key KEY shall be the KAUSF.

NOTE: FC value to be determined during normative phase.

### 6.11.3 Evaluation

TBD

## 6.12 Solution #12: Providing and Revoking Resource Owner Authorization

### 6.12.1 Introduction

This solution addresses Key Issue #2 "Checking authorization before allowing access".

This solution proposes to use a resourse owner’s 3GPP credentials for UE to generate a token, which is used to validate an API Invoker accessing the resource owner’s resources.

A resource owner in this solution is an end-user who is using the UE.

### 6.12.2 Solution details

### 6.12.2.1 Architecture



Figure 6.12.2.1-1 architecture for CAPIF with SNA enhancement

This solution uses an architecture proposed in solution #2 of TR 23.700-95 [3]. As defined in TR 23.700-95 [3], the Resource owner client is an application client used by end-user or subscriber of the API provider domain's service provider.

### 6.12.2.2 Procedure

Pre-requisite:

* During the primary authentication, AUSF receives Routing Indicator for Authorization Function from UDM.
* After the primary authentication, UE and AUSF generate S-KID (SNAAPPY Key Identifier) and KAuz from KAUSF as detailed in 6.12.2.3 and 6.12.2.4 of this document, respectively. After the key material is generated, AUSF selects the Authorization Function based on the Routing Indicator and sends the KAuz, S-KID, and SUPI to the Authorization Function. The Authorization Function stores the latest information sent by the AUSF.
* Information on Authorization Function (e.g. address, Routing Indicator, etc.) which holds the KAuz is provisioned to the UE (e.g. during registration procedure).
* API Invoker knows which APIs require a resource owner's authorization.



Figure 6.12.2.2-1 Procedure for resource owner authorization based API invocation

1. If API Invoker does not have TokenAuz for service API invocation which requires the resource owner’s authorization, API Invoker needs to request resource owner’s authorization for the API invocation even if the API invocation is authorized from API exposing function as defined in TS 33.122 [5].
2. API Invoker requests resource owner’s authorization for the API invocation. The request message includes Service API name (e.g. QoS API, location API, etc.) and API Invoker Information (e.g. API Invoker identity which is provided from CAPIF Core Function).
3. When the resource owner decides to give an authorization on the Service API name to the API Invoker (e.g. using GUI), UE generates an authorization token (TokenAuz). The claims of the TokenAuz include service API name, S-KID (SNAAPPY Key Identifier), API Invoker Information, “Authorized”, generated time, and validity time. TokenAuz contains the claims (TokenAuz, claim) and the verification information (TokenAuz, verify). Details of S-KID and the corresponding key KAuz are specified in 6.X.2.3 and 6.X.2.4 of this document, respectively. TokenAuz, verify is generated as detailed in 6.X.2.5 by using the claims and the key KAuz.
4. If the resource owner gives the authorization for the API invocation, UE responds with the TokenAuz. Upon receving the response, the API Invoker stores the TokenAuz with UE ID (e.g. application layer ID or GPSI or SUPI). The API Invoker can use the TokenAuz for the API invocation until the TokenAuz is expired by an expiration time or revoked by the resource owner, even when there is no online connection between the API Invoker and the UE.
5. API Invoker performs the service API invocation with the TokenAuz.
6. API exposing function requests token verification to Authorization Function, via CAPIF-9 interface.
7. Authorization Function finds KAuz matched to S-KID which is included in TokenAuz, claim and verifies the TokenAuz using KAuz.
8. Authorization Function responds with the verification result and UE ID (SUPI or GPSI).
9. If the verification result of the TokenAuz is successful, API exposing function stores the TokenAuz with UE ID (SUPI or GPSI). Untill API exposing function receives a revocation notification for the service API invocation or the TokenAuz is expired by an expiration time, API exposing function uses the stored TokenAuz for authorizing the API Invoker without performing token verification request to Authorization Function.
10. API Invoker receives the service API invocation response.
11. If the resource owner does not want for the API Invoker to invoke the service API, the resource owner can revoke the TokenAuz anytime before the validity time of the TokenAuz by using resource owner client. When the resource owner decides to revoke the TokenAuz for the service API, UE generates a revocation token (TokenRev). The claims of TokenRev include service API name, A-KID, API Invoker information, “Not authorized”, generated time. TokenRev contains the claims (TokenRev, claim) and the verification information (TokenRev, verify). TokenRev, verify is generated as detailed in 6.X.2.5 by using the claims and the key KAuz.
12. UE and Authorization function perform mutual authentication based on TLS-PSK as specified in clause 6.5.2.1 in TS 33.122, where PSK can be derived from KAuz. The TokenRev is transmitted to Authorization Function via CAPIF-8 interface with revocation request message.
13. Authorization Function finds KAuz by using S-KID which is included in the TokenRev, claim. Authorization Function verifies the TokenRev using KAuz.
14. Authorization Function notifies the revocation on the API Invoker’s service API invocation. The revocation notification includes the TokenRev and UE ID.
15. For the same UE ID stored at step 9 and received at step 14, API exposing function finds the TokenAuz which has same service API name and API Invoker information as the TokenRev. If the generated time of the TokenAuz is prior to that of the TokenRev, API exposing function revokes the TokenAuz and stores the TokenRev. After this, if the API Invoker performs service API invocation using the revoked TokenAuz, API exposing function shall reject the API invocation request by noticing that the generation time in the TokenAuz is prior to the generation time in the TokenRev.
16. API exposing function notifies the revocation of the token.

### 612.2.3 S-KID

S-KID is in NAI format as specified in clause 2.2 of IETF RFC 7542 [11], i.e. username@realm. The username part includes the Routing Indicator for Authorization Function and S-TID (SNAAPPY Temporary UE Identifier), and the realm part includes Home Network Identifier.

When deriving S-TID from KAUSF, the following parameters shall be used to form the input S to the KDF:

* FC = 0xXX;
* P0 = "S-TID";
* L0 = length of "S-TID";
* P1 = SUPI;
* L1 = length of SUPI.

The input key KEY shall be KAUSF.

NOTE: FC value to be determined during normative phase.

### 6.12.2.4 KAuz derivation function

When deriving KAuz from KAUSF, the following parameters shall be used to form the input S to the KDF:

* FC = 0xYY;
* P0 = “Authorization”;
* L0 = length of “Authorization”;

The input key KEY shall be the KAUSF.

NOTE: FC value to be determined during normative phase.

### 6.12.2.5 Verification information derivation

When deriving the verificiation information (TokenAuz, verify or TokenRev, verify) from KAuz, the following parameters shall be used to form the input S to the KDF.

* FC = 0xZZ;
* P0 = TokenAuz, claim or TokenRev, claim;
* L0 = length of TokenAuz, claim or TokenRev, claim;

The input key KEY shall be KAuz.

The verification information is identified with the 128 least significant bits of the output of the KDF.

NOTE: FC value to be determined during normative phase.

### 6.12.3 Evaluation

TBD

## 6.Y Solution #Y: <Title>

### 6.Y.1 Introduction

### 6.Y.2 Solution details

### 6.Y.3 Evaluation

Annex <X>:  
Change history

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
| 07-2022 | SA3#108e | S3-221597 |  |  |  | Implements S3-221314, 1359, 1660 | 0.1.0 |
| 10-2022 | SA3#108ah-e | S3-222980 |  |  |  | Implements S3-222963, 2966, 3098, 3055, 3004 | 0.2.0 |
| 11-2022 | SA3#109 | S3-224106 |  |  |  | Implements S3-224101, 3290, 4038, 4039, 4040, 4041, 4042 | 0.3.0 |
| 01-2023 | SA3#109ah-e | S3-230519 |  |  |  | Implements S3-230515, 0074, 0075, 0076, 0520, 0522, 0451, 0452, 0486, 0534, 0535, 0517, 0518, 0475, 0476, 0477, 0543, 0544, 0545 | 0.4.0 |