ETSI GS NFV-SEC 022 V4.5.1 (2024-01)

Network Functions Virtualisation (NFV) Release 4; Security; Access Token Specification for API Access

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
RGS/NFV-SEC022ed451

Keywords
API, authentication, authorization, NFV, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The common aspects for RESTful NFV MANO APIs have been defined in ETSI GS NFV-SOL 013 [22].

The ETSI NFV-MANO APIs are only allowed to be accessed by authorized consumers.

The Authorization of API Request and Authorization of notifications sending has been defined in SOL group. One solution for authorizing access is the use of OAuth with access token.

The aim of the present document is to define the Access Token for this access Authorization and associated procedure for the verification of the Access Token, ensuring security and interoperability. The present document results in a NFV profile of the OAuth2.0 for the NFV-MANO API Request and notification sending Authorization.

1 Scope

The present document defines the access tokens and related metadata for RESTful protocols and data model for ETSI NFV Management and Orchestration (MANO) interfaces. It defines also the process for the token verification by the API Producer.

For this aim, the present document:

- Analyses the security threat arising from the misuse of the access token and defines the security requirements associated to access token.
- Analyses existing specifications related to access token for API access and their compliancy with the requirements defined.
- Defines the token request and generation profile, the token format and associated metadata considering the result of existing access token specifications analysis.
- Defines the token verification procedures for the API Producer.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[2] ETSI GS NFV-SEC 002: "Network Functions Virtualisation (NFV); NFV Security; Cataloguing security features in management software".

[3] ETSI GS NFV-IFA 007: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification".

[4] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification".

[5] ETSI GS NFV-IFA 008: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Ve-Vnfm reference point - Interface and Information Model Specification".

[6] IETF RFC 6749: "The OAuth 2.0 Authorization Framework".

[7] IETF RFC 6750: "The OAuth 2.0 Authorization Framework: Bearer Token Usage".

[8] IETF RFC 7519: "JSON Web Token (JWT)".

[9] IETF RFC 3339: "Date and Time on the Internet: Timestamps".

[10] IETF RFC 7515: "JSON Web Signature (JWS)".

[11] IETF RFC 7516: "JSON Web Encryption (JWE)".

[12] NIST Special Publication 800-90B: "Recommendation for the Entropy Sources Used for Random Bit Generation", January 2018.

[13] IETF RFC 8414: "OAuth 2.0 Authorization Server Metadata".

[14] IETF RFC 7033: "WebFinger".

[15] ETSI GS NFV-IFA 011: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; VNF Descriptor and Packaging Specification".

[16] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

[17] IETF RFC 8615: "Well-Known Uniform Resource Identifiers (URIs)".

[18] IETF RFC 7591: "OAuth 2.0 Dynamic Client Registration Protocol".

[19] IETF RFC 7517: "JSON Web Key (JWK)".

[20] IETF RFC 7518: "JSON Web Algorithms (JWA)".

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

[22] ETSI GS NFV-SOL 013: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; Specification of common aspects for RESTful NFV MANO APIs".

[23] IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens".

[24] ETSI GS NFV-IFA 005: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification".

[25] ETSI GS NFV-IFA 006: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Openstack®: "All about keystone tokens".

[i.2] OpenID® Connect Core 1.0 incorporating errata set 2.

[i.3] IETF RFC 8471: "The Token Binding Protocol Version 1.0".

[i.4] IETF RFC 6819: "OAuth 2.0 Threat Model and Security Considerations".

[i.5] ETSI GS NFV-SEC 006: "Network Functions Virtualisation (NFV); Security Guide; Report on Security Aspects and Regulatory Concerns".

[i.6] ETSI TS 133 501: "5G; Security architecture and procedures for 5G System (3GPP TS 33.501)".

[i.7] draft-ietf-oauth-token-binding-08: "OAuth 2.0 Token Binding", Work in progress.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GS NFV 003 [1] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [1] and the following apply:

3GPP	3rd Generation Partnership Project
HSM	Hardware Security Module
JRD	JSON Resource Descriptor
JSON	JavaScript Object Notation
JWE	JSON Web Encryption
JWS	JSON Web Signature
JWT	JSON Web Token
MAC	Message Authentication Code
MTLS	Mutual TLS
NRF	Network Resource Function
OTP	One-Time Password
PKI	Public Key Infrastructure
REST	REpresentational State Transfer
SAML	Security Assertion Markup Language
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

4 Security requirements for API access tokens

4.1 Authorization for API access using OAuth2.0 defined in ETSI GS NFV-SOL 013

4.1.0 Authorization for API access using OAuth2.0

The requirements on interfaces supported by the reference point of MANO's entities have been defined in ETSI GS NFV-IFA 005 [24], ETSI GS NFV-IFA 006 [25], ETSI GS NFV-IFA 007 [3], ETSI GS NFV-IFA 013 [4] and ETSI GS NFV-IFA 008 [5].

One of these requirements concerns authentication and authorization of the API consumer for all operations on interfaces supported by the reference point.

To fulfil this requirement for the NFV-MANO reference points, authorization of API requests and notifications has been defined in ETSI GS NFV-SOL 013 [22].

One solution defined to handle these authorizations for API request and notification is the use of OAuth 2.0 protocol as defined by IETF RFC 6749 [6].

4.1.1 Mapping roles for Authorization for API access using OAuth2.0

For API calls, the producer functional block of an API in NFV terms corresponds to the *"resource server"*, and the consumer functional block of an API corresponds to the *"client"* as defined by IETF RFC 6749 [6]. For sending a notification, these roles are reversed: the producer (notification sender) corresponds to the *"client"*, and the consumer (notification receiver) corresponds to the *"resource server"*.

Before invoking an HTTP method on a REST resource provided by a *resource server*, a consumer functional block (referred to as "*client*" from now on) first obtains authorization from another functional block fulfilling the role of the "*authorization server*".

4.1.2 Authorization grant for Authorization for API access using OAuth2.0

Authorization grant, which is a credential representing the resource owner's authorization to access the API resources is used by the client to obtain an access token from the authorization server as defined by IETF RFC 6749 [6]. OAuth 2.0 defined 4 types of authorization grant (authorization code, implicit, resource owner password credentials, and client credentials).

For the reference points listed in clause 4.1, access to API is performed by a machine which is a non-interactive Client, acting on its own behalf and being the Resource owner. Example of such client is the EM requesting the creation of an instance of its related VNF to the corresponding VNFM; this EM is the resource owner for the management resource of the VNF. The authorization grant suitable to this case is the client credentials authorization grant. This is the authorization grant type that has been selected for the NFV-MANO interfaces and defined in ETSI GS NFV-SOL 013 [22].

4.1.3 High level procedures for API access and notifications using OAuth2.0

The roles and exchanges are shown in figure 4.1.3-1 in case of API calls and in figure 4.1.3-2 for sending a notification.

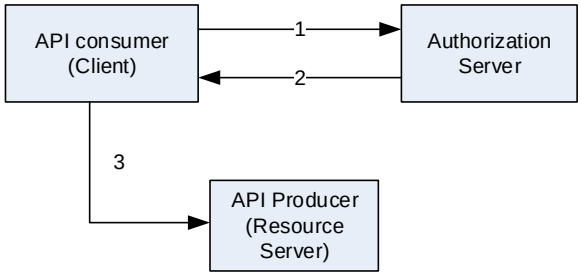

Figure 4.1.3-1: OAuth 2.0 roles in case of API calls

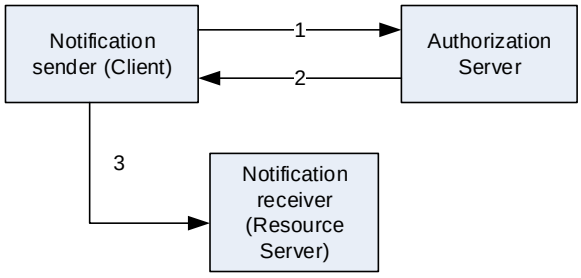

Figure 4.1.3-2: OAuth 2.0 roles in case of sending notifications

NOTE: The numbered steps below correspond to the steps of figure 4.1.3-1.

The procedure for API access is as follows:

Step 1. Before invoking the RESTful HTTP based API on the API producer, the API consumer authenticates with an Authorization Server by presenting its credentials consisting of its Client Id and Client Secret. It is assumed that authorization-related configuration parameters such as the client credentials are pre-populated in the API consumer together with other information such as the address of the *token endpoint* exposed by the *authorization server*.

Step 2. The Authorization server after authentication and validation of the API consumer returns an *access token*.

Step 3. The API consumer uses the access token in the API Request.

Same procedure is used for the notifications case shown in figure 4.1.3-2.

4.1.4 Access token for API access and notifications using OAuth2.0

An *access token* represents a particular access right (defining the particular set of protected resources to access in a particular manner) with a defined duration. The access token is usually used as a Bearer credential and transmitted in an HTTP Authorization header to the API. The token may denote an identifier used to retrieve the authorization information or may self-contain the authorization information in a verifiable manner (i.e. a token string consisting of some data and a signature). Access tokens can have different formats, structures, and methods of utilization (e.g. cryptographic properties) based on the resource server security requirements.

IETF has defined two aspects of access token use:

1) Bearer token as defined by IETF RFC 6750 [7] focuses on the transmission of the access token as an opaque string and makes no assumption about the structure of the token.

2) JSON Web Token (JWT) as defined by IETF RFC 7519 [8] focuses on the structure of the token, and allows it to be encrypted (JWE) or signed (JWS).

ETSI GS NFV-SOL 013 [22] specifies the transmission aspects of the token as a bearer token, according to the definitions by IETF RFC 6750 [7].

The present document analyses the different access token used by different standardization and open source organizations and the security threats around this access token.

4.2 Threat Analysis

4.2.0 Access token defined in ETSI GS NFV-SOL 013

The access token defined by ETSI GS NFV-SOL 013 [22] to authorize access to the API of NFV MANO interfaces is the bearer token as defined in IETF RFC 6750 [7].

The bearer token is defined in IETF RFC 6750 [7] as follows:

"Bearer Token: A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession)."

The Authorization grant type defined by ETSI GS NFV-SOL 013 [22] is the client credentials type as defined in IETF RFC 6749 [6].

4.2.1 Risk analysis and assessment

This threat analysis takes as basis the OAuth 2.0 Threat Model as presented in IETF RFC 6819 [i.4]. This risk analysis in table 4.2.1-1 uses the format found in Annex A of ETSI GS NFV-SEC 006 [i.5].

Table 4.2.1-1: Risk analysis and assessment

A Security Environment		
a.1 Assumptions		
a.1.1	It is assumed that the attacker has access to the communication between the client (API consumer) and the authorization server, and between the client (API consumer) and the resource server (API producer).	
a.1.2	An attacker has unlimited resources to mount an attack.	
a.1.3	Two of the three parties involved in the OAuth protocol may collude to mount an attack against the 3rd party.	
a.2 Assets		
a.2.1	Access token.	
a.2.2	Refresh Token.	
a.2.3	Protected Resources.	
a.2.4	Client id, client credentials.	
a.3 Threat agents		
a.3.1	Malicious authorization server: this malicious authorization server delivers bogus token and get access to client credentials or refresh token (and then obtains access token with the refresh token by counterfeiting the client).	Threats: a.4.2.2. a.4.2.3 a.4.2.4 a.4.3.3 a.4.3.6
a.3.2	Malicious client: this malicious client may modify the content of the token.	Threats: a.4.1.2
a.3.3	Attacker of client: This attack could be through malicious software within the client itself.	Threats: a.4.1.7 a.4.1.8 a.4.1.2 a.4.1.3 a.4.1.4 a.4.1.5 a.4.1.9 a.4.2.1 a.4.2.3 a.4.2.4 a.4.3.1 a.4.3.2 a.4.4.3
a.3.4	Malicious resource server: this malicious resource server gain access to the access token sent by the client by counterfeiting the resource server.	Threats: a.4.1.11 a.4.1.2 a.4.1.3 a.4.1.4 a.4.1.5 a.4.1.9 a.4.4.3 a.4.4.4

a.3.5	Malicious entity acting as a Man in the Middle on the communication between Authorization server and client.	Threats: a.4.1.1 a.4.1.2 a.4.1.3 a.4.1.4 a.4.1.5 a.4.1.9 a.4.3.4 a.4.3.6 a.4.4.1 a.4.4.3
a.3.6	Malicious entity acting as a Man in the Middle on the communication between the Client and the resource server.	Threats: a.1.4.10 a.4.1.2 a.4.1.3 a.4.1.4 a.4.1.5 a.4.1.9 a.4.4.1 a.4.4.2 a.4.4.3
a.3.7	Attacker of the Authorization server: This attack could be through malicious software within the Authorization server itself.	Threats: a.4.1.6 a.4.1.2 a.4.1.3 a.4.1.4 a.4.1.5 a.4.1.9 a.4.3.5 a.4.4.3
a.4 Threats		
a.4.1 Threats on access token		
a.4.1.1	Token Interception or token eavesdropping in transit from authorization server and client.	Mitigation by: b.1.3 b.1.4
a.4.1.2	Token Manipulation.	Mitigation by: b.1.1 b.1.18 b.1.22
a.4.1.3	Token disclosure - misuse.	Mitigation by: b.1.2 b.1.3 b.1.4 b.1.10 b.1.12 b.1.13 b.1.14 b.1.21 b.1.24
a.4.1.4	Token redirect.	b.1.3 b.1.4 b.1.10 b.1.12 b.1.13 b.1.14 b.1.21 b.1.24

a.4.1.5	Token replay.	b.1.3 b.1.4 b.1.10 b.1.12 b.1.13 b.1.14 b.1.21 b.1.24
a.4.1.6	Obtaining Access tokens from authorization server database.	b.1.4 b.1.10 b.1.12 b.1.13 b.1.14 b.1.21 b.1.24
a.4.1.7	Attacker of client obtains access tokens from the storage device.	b.1.25 b.1.9 b.1.10 b.1.12 b.1.14
a.4.1.8	Redirection on client to malicious server: Attacker of client takes the control of the client and get access to token and authorization code.	b.1.25 b.1.7 b.1.9 b.1.10 b.1.12 b.1.14
a.4.1.9	Guessing the access token.	b.1.4 b.1.10 b.1.12 b.1.13 b.1.14 b.1.17 b.1.21 b.1.24
a.4.1.10	Token Interception or token eavesdropping in the request sent from client to resource server.	b.1.4 b.1.25 b.1.10 b.1.12 b.1.13 b.1.14 b.1.16 b.1.21 b.1.24
a.4.1.11	A malicious resource server gain access to a valid access token sent by a legitimate client.	b.1.4 b.1.25 b.1.10 b.1.11 b.1.12 b.1.13 b.1.16 b.1.19 b.1.20 b.1.21 b.1.23 b.1.24

a.4.2 Threats on refresh token		
a.4.2.1	Attacker of client obtains refresh token stored in client.	b.1.6 b.1.7 b.1.9
a.4.2.2	Refresh token phishing by counterfeiting the authorization server.	b.1.4 b.1.5 b.1.6 b.1.8
a.4.2.3	Refresh token replay.	b.1.4 b.1.5 b.1.6 b.1.8
a.4.2.4	Guessing the refresh token.	b.1.26
a.4.3 Threats on client credentials		
a.4.3.1	Attacker obtains client secrets from source code.	b.1.9
a.4.3.2	Attacker obtains client secrets from a client installation.	b.1.9
a.4.3.3	malicious authorization server get access to client credentials.	b.1.15
a.4.3.4	Disclosure of client credentials during client authentication process or token requests.	b.1.27 b.1.28
a.4.3.5	Obtaining client secrets from authorization server database.	b.1.29
a.4.3.6	Guessing the client credentials.	b.1.30
a.4.4 Threats on protected resources		
a.4.4.1	An attacker eavesdrops Access tokens on transport and gain access to the protected resources.	b.1.13 b.1.21
a.4.4.2	Replay of authorized resource server requests.	b.1.16
a.4.4.3	Gain access to the protected resources by guessing the access tokens.	b.1.17
a.4.4.4	Malicious resource server gain access to a valid access token and uses it to gain access to protected resources.	b.1.19 b.1.13 b.1.21 b.1.23
B Security Objectives		
b.1 Security objectives for the asset		
b.1.1	Integrity protection of the token using digital signature or Message Authentication Code (MAC).	
b.1.2	Confidentiality protection.	
b.1.3	Access tokens should not be sent in clear over insecure channel. Use Secure transmission as TLS.	
b.1.4	Binding of the token to ID of authorized party.	
b.1.5	The authorization server should validate the client id associated with the refresh token.	
b.1.6	Revocation of refresh tokens.	
b.1.7	Revocation of client secrets.	
b.1.8	Refresh token rotation.	
b.1.9	Store secrets in secure storage.	
b.1.10	Limit token scope.	
b.1.11	Limit the token to a resource server.	
b.1.12	Limit lifetime of the access token, short access token duration.	
b.1.13	Binding of access token to client id, and client prove legitimate ownership of the token to the resource server.	
b.1.14	Allow one-time access token usage.	
b.1.15	Verification of authorization server's authenticity.	
b.1.16	Resource server uses transport security measures to avoid replay attacks (TLS) or uses signed requests with nonces and timestamps.	
b.1.17	Access token should have a reasonable level of entropy making the guessing of the token infeasible.	

b.1.18	Assertion token should be protected by a digital signature.	
b.1.19	Authentication of the resource server by the client.	
b.1.20	Bind the access token to the endpoint URL of the legitimate resource server.	
b.1.21	Binding of the access token with the client and authenticate the client with resource server requests (signature).	
b.1.22	The token contents should be protected by a digital signature.	
b.1.23	The client should authenticate the resource server before sending the access token.	
b.1.24	Authenticate the client with the resource server requests, and verification with the binding of the access token with the client id.	
b.1.25	Revocation of access token.	
b.1.26	Refresh token should have a reasonable level of entropy.	
b.1.27	The client credentials and client ID shall not be sent in clear during authentication process.	
b.1.28	The client credentials and client ID shall not be sent in clear in the token request.	
b.1.29	The authorization server database shall be a tamper resistant storage such as a HSM.	
b.1.30	The client credentials should have a reasonable level of entropy making the guessing of these credentials infeasible.	

4.3 Security requirements

From the risk analysis and assessment of clause 4.2.1, the security requirements for authentication and protocols are defined and listed in table 4.3-1.

Table 4.3-1: Requirements for authentication and protocols

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Authentication and protocols			
Auth-Prot_001	The confidentiality and data integrity of all messages shall be ensured, e.g. by using a transport-layer mechanism, such as TLS, on each interface.	a.4.1.3; a.4.1.5; a.4.2.3; a.4.3.4; a.4.4.1; a.4.4.2	
Auth-Prot_002	The client and authorization servers shall mutually authenticate.	a.4.2.2; a.4.3.3;	
Auth-Prot_003	The client shall authenticate the resource server.	a.4.4.4	
Auth-Prot_004	Before accepting the token as valid, the resource server shall authenticate the originator of the request as the legitimate owner of the token.	a.4.1.3; a.4.1.4; a.4.1.5; a.4.4.3; a.4.4.2; a.4.4.1	
Auth-Prot_005	The authorization server database used to authenticate the client and store associated client credentials, access tokens and refresh tokens shall be stored in a tamper resistant location (e.g. HSM).	a.4.3.5	

From the risk analysis and assessment of clause 4.2.1, the security requirements for client credentials are defined and listed in table 4.3-2.

Table 4.3-2: Requirements for client credentials

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Client Credentials			
Client-Cred_001	The client credentials shall be stored in a secure and tamper resistant location, or stored encrypted with the key protected in a tamper resistant location.	a.4.3.1	
Client-Cred_002	The client credentials shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources [12], in order to mitigate the risk of guessing attacks.	a.4.3.6	
Client-Cred_003	The client credentials shall not be included in the source code and software packages.	a.4.3.1	
Client-Cred_004	The client credentials shall be installed in the client in a secure way, eliminating any possibility of gaining access to these credentials during installation.	a.4.3.2	
Client-Cred_005	It shall be possible for the authorization server to revoke the client credentials.	a.4.3.1; a.4.3.2; a.4.3.3; a.4.3.4; a.4.3.5; a.4.3.6	

From the risk analysis and assessment of clause 4.2.1, the security requirements for access token are defined and listed in table 4.3-3.

Table 4.3-3: Requirements for access token

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Access token			
Acc-Token_001	The access token shall be stored in a secure and tamper resistant location or stored encrypted with the key protected in a tamper resistant location.	a.4.1.6; a.1.4.7	
Acc-Token_002	The access token shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources [12], in order to mitigate the risk of guessing attacks.	a.4.1.9; a.4.4.3	
Acc-Token_003	Access tokens shall have policy-defined limited scope.	a.4.1.1; a.4.1.3; a.4.1.4; a.4.1.5; a.4.1.6; a.4.1.7; a.4.1.8	
Acc-Token_004	Access tokens shall have limited lifetimes.	a.4.1.1; a.4.1.3; a.4.1.4; a.4.1.5; a.4.1.6; a.4.1.7; a.4.1.8; a.4.1.9; a.4.4.3	
Acc-Token_005	Access tokens shall be restricted to a particular number of operations.	a.4.1.1, a.4.1.3, a.4.1.4, a.4.1.5; a.4.1.6; a.4.1.7; a.4.1.8; a.4.1.9; a.4.4.1; a.4.4.2; a.4.4.4; a.4.4.3	
Acc-Token_006	It shall be possible to bind the access token to the intended resource server.	a.4.4.4	
Acc-Token_007	It shall be possible to bind the token to the endpoint URL (token audience) used to obtain the token.	a.4.4.4	
Acc-Token_008	It shall be possible to limit the scope of the token and associate it to particular resource.	a.4.4.1; a.4.4.2; a.4.4.3; a.4.4.4	

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Access token			
Acc-Token_009	Tokens shall be bound to the client ID.	a.4.1.1; a.4.1.3; a.4.1.4; a.4.1.5; a.4.1.6; a.4.1.7; a.4.1.8; a.4.1.9	Needs an authentication of the clients (pre-registered client-id and secret on authorization server, or secrets in the token as part of the encrypted content of the token).
Acc-Token_010	The access token shall be signed to detect manipulation of the token or production of fake tokens.	a.4.1.2	
Acc-Token_011	It shall be possible to encrypt content of the access token.	a.4.1.2; a.4.1.9	
Acc-Token_012	The access token should be defined in a standard format (SAML or JWT).		
Acc-Token_013	It shall be possible to revoke an access token.	a.4.4.1; a.4.4.2; a.4.4.3; a.4.4.4	E.g. in case of suspected compromised client.
Acc-Token_014	Unbound tokens shall not be used under any circumstance.	a.4.4.1; a.4.1.2; a.4.1.3; a.4.1.4; a.4.1.5; a.4.1.6; a.4.1.7; a.4.1.8; a.4.1.9; a.4.1.10; a.4.1.11	
Acc-Token 015	If a scheme to bind access tokens to the underlying transport layer relies on using non-standard extensions, and those extensions are not available, the system shall fail securely, preventing a bid-down attack (i.e. the resource server shall deny access).	a.4.1.10; a.4.1.11	

From the risk analysis and assessment of clause 4.2.1, the security requirements for refresh token are defined and listed in table 4.3-4.

Table 4.3-4: Requirements for refresh token

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Refresh token			
Ref-Token_001	The refresh token shall be stored in a secure and tamper resistant location or stored encrypted with the key protected in a tamper resistant location.	a.4.2.1	
Ref-Token_002	The refresh token shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources [12], in order to mitigate the risk of guessing attacks.	a.4.2.4	
Ref-Token_003	Refresh tokens shall have policy-defined limited scope.	a.4.2.1; a.4.2.2; a.4.2.3	
Ref-Token_004	Refresh tokens shall have limited lifetimes.	a.4.2.1; a.4.2.2; a.4.2.3; a.4.2.4	
Ref-Token_005	Refresh tokens shall be restricted to a particular number of operations.	a.4.2.1; a.4.2.2; a.4.2.3; a.4.2.4	

Requirement Number	Requirement Description	Reference Security threat (see clause 4.2)	Remarks
Refresh token			
Ref-Token_006	The refresh token shall be bound to the client ID.	a.4.2.1; a.4.2.2; a.4.2.3; a.4.2.4	Needs an authentication of the clients (pre-registered client-id and secret on authorization server, or secrets in the token as part of the encrypted content of the token).
Ref-Token_007	It shall be possible to rotate refresh tokens by changing the value of the refresh token with every refresh request.	a.4.2.1; a.4.2.2; a.4.2.3; a.4.2.4	
Ref-Token_008	It shall be possible to revoke a refresh token.	a.4.2.1; a.4.2.2; a.4.2.3; a.4.2.4	In case of suspected compromised client.

5 NFV Access Token Definition

5.1 Authorization Server discovery

5.1.1 Authorization Server discovery description

As described in clause 8.2 of ETSI GS NFV-SOL 013 [22], before invoking an HTTP method on a REST resource provided by a *resource server*, a functional block (referred to as "*client*") first obtains authorization from another functional block fulfilling the role of the "*authorization server*".

The first step for a consumer of an API is then to connect to the appropriate authorization server managing the authorization for the corresponding API producer to obtain the access token.

The API consumer needs to know which authorization server is relevant for the authorization request and needs to know the address of the token endpoint exposed by this Authorization Server and its associated public key to establish a secure channel with this endpoint. For this purpose an Authorization Server discovery is processed by the API consumer as described in IETF RFC 8414 [13]. During this process the API consumer retrieves the metadata for the authorization server from a well-known location as a JSON document, which declares its endpoint locations (e.g. authorization endpoint, token endpoint, registration endpoint) and authorization server capabilities, e.g. its capability to support MTLS as described in [23]. This well-known location is based on the Authorization Server Identifier as defined in clause 5.1.4. The Authorization Server Identifier is discovered as described below.

The API consumer may either discover the Authorization Server Identifier (Issuer Identifier) using a pre-configuration or may dynamically discover it through the use of WebFinger IETF RFC 7033 [14] as described in IETF RFC 8414 [13].

The first approach is typically used when the Authorization server for the API Consumer is well-known and static. The Identifier of the Authorization Server to access the VNFM for a VNFI/VNFCI is static but usually not known at the VNF packaging time. A configuration of the authorization server identifier in the VNFI/VNFCI may be done during the instantiation process and any time after the instantiation via configuration.

For the case where multiple instances of MANO entities are dynamically deployed, a dynamic discovery of the authorization server identifier might be better fit to purpose.

5.1.2 Manual Authorization Server Identifier discovery

For this manual Authorization Server Identifier discovery, the Issuer Identifier is statically configured in the API Consumer.

If the API Consumer is a VNF instance, the following mechanism can be used to provide a VNF instance with an authorization server identifier associated to the VNFM, to access VNFM APIs:

- A VNF configurable property as described in clause 5.8 of ETSI GS NFV-IFA 011 [15] is declared in the VNFD. The value of this VNF configurable property is provided by the VNFM to the VNF instance during the VNF instantiation. The value can be modified any time after instantiation.

If the API Consumer is a MANO entity, the Authorization Server Identifier is statically configured in this MANO entity during the deployment process, by means out of the scope of the present document.

If the API Consumer is a dynamically deployed VNF-specific EM, the Authorization Server Identifier is statically configured in this entity during the deployment process, by means out of the scope of the present version of the present document.

5.1.3 Dynamic Authorization Server Identifier discovery

NOTE: It is out of scope of the present document how trust with the Authorization Server for dynamic discovery is established.

The dynamic discovery of Authorization Server is done through the use of WebFinger IETF RFC 7033 [14] as described in IETF RFC 8414 [13]. Dynamic discovery of Authorization server is optional. If the API consumer knows the API Producer's Authorization Server Identifier through an out-of-band mechanism, it can skip this step.

The WebFinger request is an HTTPS request to WebFinger resource which is a "well-known" URI using the HTTPS scheme. The path of a WebFinger URI is:

"/.well-known/webfinger"

The request includes the following parameters:

- Resource: Identifier for the API producer that is the subject of the discovery request.
- Host: Server where a WebFinger service is hosted.
- Rel: URI identifying the type of service whose location is being requested.

The following "rel" value for the discovery of the OAuth Authorization Server Identifier in Webfinger is defined:

Rel Type	URI
ETSI NFV OAuth Authorization Server Identifier	urn:etsi:nfv:webfinger:rel:api-oauth-server

In case of VNF discovering the OAuth Authorization server for an API exposed by VNFM that manages it, the "resource" value is the URI prefix of the API exposed by the VNFM (i.e. the sequence of {apiRoot}/{apiName}/{apiMajorVersion} as defined in clause 4.1 of ETSI GS NFV-SOL 013 [22]) and the "host" value is the server where the WebFinger service is hosted. The API URI prefixes and the host which provides the Webfinger resource are provisioned into the VNF instance by the VNFM during the VNF instantiation. Clause 5.8 of ETSI GS NFV-IFA 011 [15] defines VNF configurable properties for this purpose.

The WebFinger resource returns a JSON Resource Descriptor (JRD) as defined in IETF RFC 7033 [14] to convey information about the Authorization Server controlling the access to the resources provided by the API producer, containing the Authorization Server Identifier. The Identifier is in form of URI as defined in IETF RFC 3986 [16] with a scheme component that shall be HTTPS, a host component and optionally, port and path components and no query or fragment components.

The OAuth Authorization Server Identifier is included in the "links" array as the value of the "href" corresponding to the link's relation type "rel": urn:etsi:nfv:webfinger:rel:api-oauth-server as described in IETF RFC 7033 [14].

EXAMPLE: Discovery of the Authorization Server for the VNFM managing a specific VNF.

In this example the API prefix of the VNF LCM API produced by the VNFM is given in form of URL Syntax:

- Resource: https://vnfm.example.com/someprefix/vnflcm/v1
- Rel: urn:etsi:nfv:webfinger:rel:api-oauth-server
- Host: webfinger.example.com

```
GET /.well-known/webfinger
    ?resource=https%3A%2F%2Fvnfm.example.com%2Fsomeprefix%2Fvnflcm2F%v1
&rel=urn%3Aetsi%3Anfv%3Awebfinger%3Arel%3Aapi-oauth-server
    HTTP/1.1
  Host: webfinger.example.com

  HTTP/1.1 200 OK
  Content-Type: application/jrd+json

  {
   "subject": "https://vnfm.example.com/someprefix/vnflcm/v1",
   "links":
    [
     {
      "rel": "urn:etsi:nfv:webfinger:rel:api-oauth-server",
      "href": "https://oauth.server.example.com"
     }
    ]
  }

     }
    ]
  }
```

5.1.4 Authorization Server Configuration discovery

Using the Authorization Server Identifier discovered as described in clause 5.1.2 or 5.1.3, the Authorization Server configuration information can be retrieved.

An Authorization Server Configuration Document (JSON document) shall be queried using an HTTP GET request via HTTPS to the URI formed by inserting the well-known URI string defined for ETSI NFV "/.well-known/nfv-oauth-server-configuration" into the URI that represents the authorization server identifier after the "host" component and optional "port" component and before the optional "path" component, as defined in section 3 of IETF RFC 8414 [13].

EXAMPLE 1: Get request in case the Authorization Server Identifier is https://example.com:

```
 GET /.well-known/ nfv-oauth-server-configuration HTTP/1.1
 Host: example.com
```

EXAMPLE 2: Get request in case the Authorization Server Identifier contains a path component https://example.com/issuer1:

```
GET /.well-known/ nfv-oauth-server-configuration/issuer1 HTTP/1.1
 Host: example.com
```

The related HTTP response shall contain a payload body formatted as a JSON document compliant with the provisions defined in table 5.1.4-1 of the present document and shall contain a Content-Type header set to "application/json".

The following metadata in table 5.1.4-1 are defined for this JSON document to describe the Authorization Server configuration:

Table 5.1.4-1: Authorization server configuration JSON metadata

Claim	Qualifier	Cardinality	Content	Description
issuer	M	1	String	URL using the https scheme with no query or fragment component that the Authorization Server asserts as its Issuer Identifier. If dynamic discovery of Authorization Server as described in clause 5.1.3 is used, this value shall be identical to the issuer value returned by WebFinger. This also shall be identical to the iss Claim value in NFV Tokens issued from this Issuer.
authorization_endpoint	O	0..1	String	URL of the OAuth 2.0 Authorization Endpoint.
token_endpoint	M	1	String	URL of the OAuth 2.0 Token Endpoint.
jwks_uri	M	1	String	URL of the JSON Web Key Set as defined in IETF RFC 7517 [19].
registration_endpoint	O	0..1	String	URL of the Dynamic Client Registration Endpoint.

Claim	Qualifier	Cardinality	Content	Description
scopes_supported	O	0..N	String	JSON array containing a list of the OAuth 2.0 scope values as defined in IETF RFC 6749 [6] that this server supports.
response_types_supported	M	1..N	String	JSON array containing a list of the OAuth 2.0 response_type values that this server supports. NFV Authorization Server shall support the "token nfv_token" Response Type values.
grant_types_supported	M	1..N	String	JSON array containing a list of the OAuth 2.0 Grant Type values that this server supports. NFV Authorization Server shall support the "client_credentials" and may support other Grant Types.
nfv_token_signing_alg_values_supported	M	1..N	String	JSON array containing a list of the JWS signing algorithms (alg values as defined in IETF RFC 7515 [10]) supported by the server for the NFV access token to encode the Claims in a JWT. The "RS256" alg value shall always be included. The value "none" shall not be used.
nfv_token_encryption_alg_values_supported	O	0..N	String	JSON array containing a list of the JWE encryption algorithms (alg values) supported by the server for the NFV access token to encode the Claims in a JWT as defined in IETF RFC 7519 [8].
nfv_token_encryption_enc_values_supported	O	0..N	String	JSON array containing a list of the JWE encryption algorithms (enc values) supported by the server for the NFV access token to encode the Claims in a JWT as defined in IETF RFC 7519 [8].
token_endpoint_auth_methods_supported	O	0..N	String	JSON array containing a list of the JWS signing algorithms (alg values as defined in IETF RFC 7515 [10]) supported by the Token Endpoint for the signature on the JWT used to authenticate the Client at the Token Endpoint for the private_key_jwt and client_secret_jwt authentication methods. Servers should support "RS256". The value "none" shall not be used. If Mutual TLS Authentication as defined in IETF RFC 8705 [23] is used the following methods may be supported: "tls_client_auth" if PKI methods is used and "self_signed_tls_client_auth" if Self-Signed Certificate method is used.
token_endpoint_auth_signing_alg_values_supported	O	0..N	String	JSON array containing a list of the JWS signing algorithms (alg values) supported by the Token Endpoint for the signature on the JWT used to authenticate the Client at the Token Endpoint for the private_key_jwt and client_secret_jwt authentication methods. Servers should support "RS256". The value "none" shall not be used.
tls_client_certificate_bound_access_tokens	M	1	Boolean	Boolean value indicating server support for mutual TLS as defined in IETF RFC 8705 [23] client certificate bound access tokens. Authorization Server shall support mutual TLS and the Boolean value shall be "true".
claims_supported	O	0..N		JSON array containing a list of the Claim Names of the Claims that the server is able to supply values for.
require_request_uri_registration	O	0..1	Boolean	Boolean value specifying whether the server requires any request_uri values used to be pre-registered using the request_uris registration parameter. Pre-registration is required when the value is "true". If omitted, the default value is "false".

The response of the Authorization Server Configuration request is a set of JWT Claims (as defined in IETF RFC 7519 [8]) about the Authorization Server configuration, including all necessary endpoints and public key location

information. A successful response shall use the 200 OK HTTP status code and return a JSON object using the application/json content type that contains a set of Claims as its members that are a subset of the Metadata values defined above. Other JWT Claims may also be returned. Claims that return multiple values are represented as JSON arrays. Claims with zero elements shall be omitted from the response.

An error response uses the applicable HTTP status code value.

5.2 Registration process

5.2.1 Disposition

The description of the registration process in the entire clause 5.2 is for further study and presented only for information.

Throughout the present document, the term "NFV access token" is used as a shorthand for an access token that conforms to the provisions in clause 5.5 of the present document.

5.2.2 Registration process description

The Authorization server maintains the profile of each entity for which it controls the API access, including e.g. the type of the entity, the access policies for the API with some other parameters. This profile is created during the registration process as described in section 2 of IETF RFC 6749 [6]. During this process, the authorization server issued the registered client a client identifier as described in section 2 of IETF RFC 6749 [6].

The registration of the client with the authorization server is done as described in IETF RFC 7591 [18]. To register the client with the authorization server, an HTTP POST is sent to the client registration endpoint with a content type of "application/json". The HTTP Entity Payload is a JSON document containing a software statement includes in the JSON object using the software_statement member as described in IETF RFC 7591 [18].

The software_statement member contains all client metadata values about the client software as claims. This is a string value containing the entire signed JWT.

The registration process for a VNF instance and its interaction with the VNF instantiation process require further study and are thus outside the scope of the present version of the present document.

5.2.3 Client metadata

The Client metadata are defined in table 5.2.3-1 and are included in the software statement as a JSON Web Token (JWT) as defined in IETF RFC 7519 [8]. The software statement is signed using digital signatures or Message Authentication Codes (MAC) based on JSON Web Signature (JWS) as defined in IETF RFC 7515 [10].

Table 5.2.3-1: Client metadata of the software statement

Claim	Qualifier	Cardinality	Content	Description
token_endpoint_auth_method	M	1	String	String indicator of the requested authentication method for the token endpoint. The value "none" is not used. If Mutual TLS Authentication as defined in IETF RFC 8705 [23] is used the following methods may be requested: "tls_client_auth" if PKI methods is used and "self_signed_tls_client_auth" if Self-Signed Certificate method is used.
grant_types	M	1	String	"client_credentials". Other values defined in IANA for "grant_types" are not used.
response_types	M	1	String	"none". Other value defined in IANA for "response_type" are not used.

Claim	Qualifier	Cardinality	Content	Description
scope	O	0..1	String	String containing a space-separated list of scope values (as described in section 3.3 of IETF RFC 6749 [6]) that the client can use when requesting access tokens. The semantics of values in this list are service specific. If omitted, an authorization server may register a client with a default set of scopes.
jwks_uri	M	1	String	URL of the client's JSON Web Key Set as described in IETF RFC 7517 [19], which contains the client's public keys.
software_id	M	1	String	A unique identifier string (e.g. a Universally Unique Identifier (UUID)) assigned by the client developer or software publisher used by registration endpoints to identify the client software to be dynamically registered. Unlike "client_id", which is issued by the authorization server and should vary between instances, the "software_id" should remain the same for all instances of the client software. The "software_id" should remain the same across multiple updates or versions of the same piece of software. The value of this field is not intended to be human readable and is usually opaque to the client and authorization server.
software_version	M	1	String	A version identifier string for the client software identified by "software_id".
nfv_token_signed_response_alg	O	0..1	String	JWS alg algorithm as defined in IETF RFC 7518 [20] for signing the NFV access token issued to this Client. The value none is not used as the NFV access token alg value. The default, if omitted, is RS256. The public key for validating the signature is provided by retrieving the JWK Set referenced by the jwks_uri element.
nfv_token_encrypted_response_alg	O	0..1	String	JWE alg algorithm as defined in IETF RFC 7518 [20] for encrypting the NFV access token issued to this Client. If this is requested, the response will be signed then encrypted, with the result being a Nested JWT, as defined in IETF RFC 7519 [8]. The default, if omitted, is that no encryption is performed.
nfv_token_encrypted_response_enc	O	0..1	String	JWE enc algorithm as defined in IETF RFC 7518 [20] for encrypting the NFV access token issued to this Client. If nfv_token_encrypted_response_alg is specified, the default for this value is A128CBC-HS256. When nfv_token_encrypted_response_enc is included, nfv_token_encrypted_response_alg is also provided.
token_endpoint_auth_signing_alg	O	0..N	String	JWS [10] alg algorithm as defined in IETF RFC 7518 [20] that is used for signing the JWT [8] used to authenticate the Client at the Token Endpoint for the private_key_jwt and client_secret_jwt authentication methods. All Token Requests using these authentication methods from this Client are rejected, if the JWT is not signed with this algorithm. Servers should support RS256. The value none is not used. The default, if omitted, is that any algorithm supported by the Authorization server token endpoint may be used.

Claim	Qualifier	Cardinality	Content	Description
tls_client_certificate_bound_access_tokens	M	1	Boolean	Indicates the client's intention to use mutual TLS client certificate bound access tokens as defined in IETF RFC 8705 [23]. For the present version of the present document the value shall be "true".
tls_client_auth_subject_dn	CM	0..1	String	String value specifying the expected subject distinguished name of the client certificate, which the OAuth client will use in mutual TLS authentication. The presence of this claim is mandatory if "tls_client_certificate_bound_access_tokens "value is "true".

5.3 Token Request

In clause 4.3, Auth-Prot_001, Auth-Prot_002, Auth-Prot_003 and Auth-Prot_004 requirements are defined for the protocols between authorization server and client and between client and resource server.

The use of OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens as defined in IETF RFC 8705 [23] allows to fulfil all these requirements.

Auth-Prot_001 and Auth-Prot_002 are fulfilled by the Mutual TLS Client Authentication part described in section 2 of IETF RFC 8705 [23].

The Auth-Prot_003 and Auth-Prot_004 are fulfilled by the Mutual TLS Client Certificate Bound Access Tokens part described in section 3 of IETF RFC 8705 [23]. This method ensures that only the party in possession of the private key corresponding to the certificate can utilize the token to access the associated resources.

The use of OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens as defined in IETF RFC 8705 [23] is mandatory for API access control based on OAuth2.0. Two methods of authentication may be used: PKI Mutual TLS OAuth Client Authentication Method, or Self-Signed Certificate Mutual TLS OAuth Client Authentication Method. The authentication method used is declared in "token_endpoint_auth_method" metadata during the registration process as described in clause 5.2.

The TLS connection between the client and the authorization server token endpoint shall be established with mutual TLS X.509 certificate authentication, i.e. using certificate and certificate verify messages sent during the TLS Handshake.

The client shall include in all requests to the authorization server, the "client_id" parameter, configured in the client after the registration process as described in clause 5.2.

The API Consumer makes a Token Request by presenting its Client Credentials to the Token Endpoint using the "grant_type" value "client_credentials", as described in section 4.4.2 of IETF RFC 6749 [6]. The Client sends the parameters to the Token Endpoint using the HTTP POST method and the Form Serialization, as described in section 4.4.2 of IETF RFC 6749 [6]. The URL of the Token Endpoint is retrieved in the Authorization server configuration as defined in clause 5.1.4.

The Authorization Server authenticates the API consumer and if valid, issues the NFV access token as JWT access token as defined in IETF RFC 7519: "JSON Web Token (JWT)" [8], with the claims defined in clause 5.5.

To bind the certificate to the access token, the hash of the certificate is included by the authorization server in the "x5t#S256" confirmation method of the NFV access token as described in clause 5.5.

The API consumer includes in the API requests to the API Producer (i.e. in requests to protected resources), the NFV access token in the Authorization request header field, as described in IETF RFC 6750 [7].

These requests shall be made over a mutually authenticated TLS connection using the same client certificate that was used for mutual TLS at the token endpoint.

EXAMPLE: API consumer request to API producer including the NFV access token (with line breaks for display purposes only).

```
GET /resource HTTP/1.1
```

```
Host: server.example.com
Authorization: Bearer eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQdBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

The protected resource verifies that the client certificate matches the certificate associated with the NFV access token. If they do not match, the resource access is rejected with an error as defined in IETF RFC 8705 [23].

Mutual TLS is used only as a proof-of-possession mechanism during protected resource access. The resource server should therefore configure the TLS stack in a way that it does not verify whether the certificate presented by the client during the handshake is signed by a trusted CA certificate.

5.4 NFV Access Token Format

The NFV access token to authorize access to the API of NFV-MANO interfaces is transmitted, according to ETSI GS NFV-SOL 013 [22], like a bearer token as defined in IETF RFC 6750 [7].

It is emphasized that the NFV access token as defined in the present document is strictly speaking not a bearer token as defined in IETF RFC 6750 [7], as it adds the property to give a proof-of-possession during protected resource access, i.e. it cannot be used by any party in possession of this token, but only by the legitimate owner. However, the NFV access token is transmitted using the protocol defined for bearer token in IETF RFC 6750 [7].

To fulfil the requirements from Acc-Token_003 to Acc-Token_009 and Acc-Token_014 in clause 4.3, the NFV access token shall be associated to different parameters to restrict the lifetime, the number of operations, to bind the token to a clientID, etc.

The NFV access token shall then be associated to some metadata. These metadata are defined in clause 5.5.

In clause 4.3, the requirement Acc-Token_012 recommends a standard format for the definition of NFV access token.

For the defined NFV-MANO interfaces, where OAuth2.0 is used for API access control, NFV access token is defined as a JSON Web Token as described in IETF RFC 7519: "JSON Web Token (JWT)" [8], including claims defined in clause 5.5 that bind this NFV access token to the TLS client certificate of the API consumer that receives the NFV access token from the Authorization server, and restrict the use of the NFV access token.

The NFV access token shall be included in the token response (as defined by IETF RFC 6749 [6]).

In clause 4.3, the Acc-Token_010 requires that the NFV access token is signed.

To ensure the integrity of the NFV access token and to authenticate the issuer, the JWT document shall be signed using digital signatures or Message Authentication Codes (MAC) based on JSON Web Signature (JWS) as described in IETF RFC 7515 [10].

In clause 4.3, the Acc-Token_011 requires the possibility to encrypt the content of NFV access token.

The NFV access token JWT document shall be signed and then may be encrypted using JSON Web Signature (JWS) as defined in IETF RFC 7515 [10] and JSON Web Encryption (JWE) as defined in IETF RFC 7516 [11] respectively.

5.5 NFV access token associated Metadata

The following claims in table 5.5-1 are defined for the NFV access token:

Table 5.5-1: NFV access token claims

Claim	Method	Qualifier	Cardinality	Content	Description	Associated requirement
iss		M	1	String	Issuer Identifier for the Issuer of the token. The iss value is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.	
sub		M	1	String	Subject Identifier. A locally unique and never reassigned identifier within the Issuer for API producer which provide protected resources consumed by the client. It shall not exceed 255 ASCII characters in length. The sub value is a case sensitive string.	Acc-Token_006 Acc-Token_008
aud		M	1	String	Audience(s) that this NFV Token is intended for. It shall contain the OAuth 2.0 "client_id" as an audience value. The "aud" value is a case sensitive string.	Acc-Token_009, Acc-Token_007
exp		M	1	JSON number	Expiration time on or after which the NFV Token shall not be accepted for processing. The processing of this parameter requires that the current date/time shall be before the expiration date/time listed in the value. Implementers may provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. See IETF RFC 3339 [9] for details regarding date/times in general and UTC in particular.	Acc-Token_004
iat		M	1	JSON number	Time at which the JWT was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.	Acc-Token_004
auth_time		CM	0..1	JSON number	Time when the client authentication occurred. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time. When "auth_time" is requested in the token request as an Essential Claim using "claims" parameter, then this Claim is mandatory.	
jti		M	1	String	Unique identifier for the JWT. The identifier value shall 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 and that the value is not guessable; and collisions shall be prevented among values produced by different issuers as well. The "jti" claim is used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string.	
cnf	x5t#S256	M	1	String	Hash of the client certificate used for the mutual TLS at the token endpoint as described in IETF RFC 8705 [23]. Value is a base64url-encoded SHA-256 hash (i.e. thumbprint, fingerprint or digest) of the DER encoding of the X.509 certificate. The base64url-encoded value shall omit all trailing pad '=' characters and shall not include any line breaks, whitespace, or other additional characters.	Acc-Token_009
scope		M	0..1	String	String containing a space-separated list of scope of operation values (as described in section 3.3 of IETF RFC 6749 [6]) for which the NFV access token is valid. The semantics of values in this list are service specific. If the value is omitted, it means that the NFV access token is valid for all API operations. Other values are defined in the corresponding API specification.	Acc-Token_003 Acc-Token_008

Claim	Method	Qualifier	Cardinality	Content	Description	Associated requirement
at_use_nbr		M	1	JSON number	Non-negative integer that represents the number of API requests for which the NFV access token can be used. If the value is set to 0, the NFV access token can be used until it expires, as defined by the "exp" claim. See note.	Acc-Token_005
NOTE: The "at_use_nbr" claim improves the security of the NFV access token as it helps limiting the effect of attacks that extend the expiry time of an NFV access token. API producers should support non-zero values of the "at_use_nbr" claim (i.e. they should be able to count in a secure way how often an NFV access token was used). API producers that do not support non-zero values of the "at_use_nbr" claim shall reject a token with non-zero values of this claim.						

6 Token Verification Process

The API producer verifies the NFV access token presented by the API consumer before answering to the API consumer requests. It uses the token information included in the NFV access token for its verification.

It verifies among others, the validity of the NFV access token, and the scope value for which the NFV access token was issued. It shall authenticate the originator of the request as the legitimate owner of the token, verifying that the hash value of the certificate in the NFV access token is the same as the hash value of the client certificate used for the mutual TLS establishment with the API consumer.

The resource server may also use the OAuth2.0 Token Introspection method as defined in IETF RFC 7662 [21] to get from the authorization server, the metadata associated to the NFV access token.

Annex A (informative): Analysis of existing Access Token specifications

A.1 OpenStack® Keystone

A.1.0 Introduction

Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API.

In OpenStack Keystone, the tokens are used to authenticate and authorize interactions with the various OpenStack APIs. Tokens come in many scopes, representing various authorization and sources of identity.

NOTE: Information in this clause gives high level description of OpenStack Keystone token got from the keystone website [i.1] and is for information only.

A.1.1 Authorization scopes

Several authorization scopes are possible for the tokens:

- Unscoped tokens: An unscoped token contains neither a service catalogue, any roles, a project scope, nor a domain scope. Their primary use case is simply to prove your identity to keystone at a later time (usually to generate scoped tokens), without repeatedly presenting your original credentials.
- Project-scoped tokens: They contain a service catalogue, a set of roles, and details of the project upon which the tenant of the token has authorization.
- Domain-scoped tokens: They contain a limited service catalogue (only those services which do not explicitly require per-project endpoints), a set of roles, and details of the project upon which the tenant of the token has authorization. They express authorization to operate a domain-level, typically as a domain-level administrator.
- System-scoped tokens: It represents the role assignments a user has to operate on the deployment as a whole, i.e. for operations that affect the entire deployment system such as e.g. modifying endpoints, service management, or listing information about hypervisors.

A.1.2 Token binding

OpenStack Keystone may support token binding. In the OpenStack Newton release, there are four supported token types: UUID, PKI, PKIZ and fernet. Since the OpenStack Ocata release, PKI and PKIZ tokens are deprecated. In the Pike release, UUID token are also deprecated and the fernet token was the default and only supported token type. As of the Stein release support was added for JSON Web Signature tokens. UUID token, PKI and PKIZ tokens are described in ETSI GS NFV-SEC 002 [2] for further information on these token types.

Token binding embeds information from an external authentication mechanism, such as a Kerberos server or X.509 certificate, inside a token. By using token binding, a client can enforce the use of a specified external authentication mechanism with the token. This additional security mechanism ensures that if a token is stolen, for example, it is not usable without external authentication.

A.1.3 Fernet token

The default token format supported by keystone is the fernet token. Fernet tokens are bearer token. They are protected from unnecessary disclosure to prevent unauthorized access.

Fernet tokens do not need to be persistent in a back end. AES256 encryption is used to protect the information stored in the token and integrity is verified with a SHA256 HMAC signature. Only the Identity service should have access to the keys used to encrypt and decrypt fernet tokens. Like UUID tokens, fernet tokens are passed back to the Identity service in order to validate them.

The binding to other attributes as an additional authentication method (e.g. kerberos or x509certificate) is not supported by fernet token.

OAuth access tokens can be exchanged for keystone tokens.

The validation of the token can be done only on-line. An offline validation, i.e. Self-validation of tokens, rather than calling back to keystone, in order to improve performance and scalability, is not supported by fernet tokens.

Fernet tokens contain a limited amount of identity and authorization data in a MessagePacked payload. The data inside a fernet token is protected using symmetric encryption keys, or fernet keys.

A.1.4 Fernet keys

A fernet key is used to encrypt and decrypt fernet tokens. Each key is actually composed of two smaller keys: a 128-bit AES encryption key and a 128-bit SHA256 HMAC signing key.

The keys are held in a key repository that keystone passes to a library that handles the encryption and decryption of tokens. A key repository is required by keystone in order to create fernet tokens. These keys are used to encrypt and decrypt the information that makes up the payload of the token.

Each key in the repository can have one of three states: primary key, secondary key and staged key.

Each key starts as a staged key, is promoted to be the primary key, and then demoted to be a secondary key.

New tokens can only be encrypted with a primary key. Secondary and staged keys are never used to encrypt token.

The staged key is used to perform a key rotation on one keystone node, and distribute the new key set over a span of time. This does not require the distribution to take place in an ultra-short period of time. Tokens encrypted with a primary key can be decrypted, and validated, on other nodes where that key is still staged.

A.1.5 Advantage of Fernet tokens

Fernet tokens, unlike UUID tokens, do not require persistence and do not have to be replicated. As long as each keystone node shares the same key repository, the fernet tokens can be created and validated instantly across nodes.

In addition the advantage of fernet tokens over PKI or PKIZ tokens is the fact that the fernet tokens are much smaller. The fernet tokens are kept under 250 byte limit.

A.1.6 JSON Web Signature token

The JSON Web Signature (JWS) token format is a type of JSON Web Token (JWT) as defined by IETF RFC 7519 [8]. JWS tokens are ephemeral, base64 encoded and digitally signed. Therefore it is essential not to include sensitive information in the token. JWS tokens use asymmetric keys for signing, where private keys sign the tokens, and public keys validate them. In OpenStack Stein and later versions, the JWS token provider implementation exclusively supports the ES256 JSON Web Algorithm (JWA).

A.1.7 JSON Web Signature keys

JWS tokens are signed use ES256 JSON Web Algorithm (JWA), which employs the Elliptic Curve Digital Signature Algorithm (ECDSA), the P-256 curve, and a SHA-256 hash algorithm. More information about JWA can be found in the IETF specification IETF RFC 7518 [20].

To ensure proper functionality, it is necessary to synchronize public keys across all OpenStack keystone nodes in a deployment. Each keystone server should have a corresponding public key for every node. This only applies to public keys, private keys should never leave the server where they are generated. The private key used to sign JWS tokens are stored in the JWS private key repository as configured in keystone.

A.1.8 Advantage of JSON Web Signature token

When deploying OpenStack using JWS tokens as opposed to fernet tokens may be considered if there are security concerns about sharing symmetric encryption keys across hosts. The primary benefit of using asymmetric keys in OpenStack is that each keystone server generates its own key pair. The private key signs tokens, and anyone with access to the public key can verify the token signature. This is a critical step in validating tokens across a cluster of keystone nodes.

However it is essential to note a major difference between Fernet and JWS tokens. JWS tokens are not encrypted and can be decoded by anyone with the token ID Whereas Fernet tokens encrypt the ID with a symmetric key for added security.

A.2 OpenID® Connect ID-Token

A.2.0 Introduction

OpenID connect is an identity layer on the top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

The Authorization server (called the OpenID Provider) answers to the Client AuthN Request with an ID token and usually an Access Token.

The OAuth2.0 token_type response parameter value is "bearer" and in addition to the response parameters specified by OAuth2.0, the id_token is included in the response.

NOTE: Information in this clause gives high level description of OpenID Connect ID-Token got from the OpenID Connect website [i.2] and is for information only.

A.2.1 ID Token

The primary extension of the OpenID Connect is the use of ID Token, which is a security token containing claims about the Authentication of the End-User by the Authorization Server, and potentially other claims. The Authentication result is returned in an ID Token using claims expressing such information as the Issuer, the Subject Identifier, when the authentication expires, etc.

The ID Token is represented as a JSON Web Token (JWT) as defined by IETF RFC 7519 [8].

Claims used in the ID Token for all OAuth 2.0 flows used by OpenID Connect are described in clause 2 ID Token of [i.2] and are described in table A.2.1-1.

Table A.2.1-1: Claims used for OpenID Connect ID Token

Claim	Description
iss	Issuer Identifier for the Issuer of the response.
sub	Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client.
aud	Audience that this ID Token is intended for. It contains the OAuth 2.0 client_id of the Relying Party as an audience value. It may also contain identifiers for other audiences.
exp	Expiration time on or after which the ID Token is not be accepted for processing. The processing of this parameter requires that the current date/time is before the expiration date/time listed in the value.
iat	Time at which the JWT was issued.
auth_time	Time when the End-User authentication occurred.
nonce	String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. If present in the ID Token, Clients verify that the nonce Claim Value is equal to the value of the nonce parameter sent in the Authentication Request. If present in the Authentication Request, Authorization Servers include a nonce Claim in the ID Token with the Claim Value being the nonce value sent in the Authentication Request. Authorization Servers performs no other processing on nonce values used.
acr	Authentication Context Class Reference. String specifying an Authentication Context Class Reference value that identifies the Authentication Context Class that the authentication performed satisfied.
amr	Authentication Methods References. JSON array of strings that are identifiers for authentication methods used in the authentication (e.g. password and OTP authentication methods).
azp	Authorized party - the party to which the ID Token was issued. If present, it contains the OAuth 2.0 Client ID of this party. This Claim is only needed when the ID Token has a single audience value and that audience is different than the authorized party.
at_hash	Access Token hash value.
c_hash	Code hash value. Used in case of hybrid flow, where an authorization code is used.

ID Tokens may contain other claims.

ID Token are signed using JSON Web Signature (JWS) as defined in IETF RFC 7515 [10] and optionally signed and then encrypted using JSON Web Signature (JWS) as defined in IETF RFC 7515 [10] and JSON Web Encryption (JWE) as defined in IETF RFC 7516 [11] respectively.

A.2.2 Advantage of ID Token

ID Token embeds information from authentication mechanism used to authenticate the End_User. The ID Token is then used as token binding. This additional security mechanism ensures that if the token is stolen, for example, it is not usable without external authentication.

The ID Token is bound also to the access token using the at_hash claim. The access_token is then in its turn bound to the authentication mechanism and cannot be used without such authentication.

This binding avoid the use of a stolen access token.

An additional advantage of the ID Token is the use of additional claims that enables the mitigation of replay attacks such as the "nonce" claim, or enabling a lifetime for the token such as the "exp" claim, or defining the expected audience such as the "aud" claim, or defining the issuer of the token such as the "iss" claim.

Additional claims may be defined to add other bindings if needed.

The id_token is defined in IANA and may be used with the bearer access token in the response by the token end-point, without any change of the OAuth2.0 protocol.

A.3 IETF TLS-Based AccessToken Binding

A.3.0 Introduction

In general, any party in possession of bearer security tokens gain access to certain protected resource. Attackers take advantage of this by exporting bearer tokens from a Client and presenting them to application servers, and impersonating authenticated users.

The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the underlying TLS layer and then to scope the applicability of the access token to a certain sender. The sender is then obliged to demonstrate knowledge of a certain secret as pre-requisite for the acceptance of the access token by the resource server.

IETF is currently defining this kind of token binding to access tokens, authorization codes or grants and client authentication in "OAuth 2.0 Token Binding" as defined in draft-ietf-oauth-token-binding [i.7]. This use of token binding protects token from man-in-the-middle and token export and replay attacks. The access token is, via a token binding id, finally bound to the TLS connection used between the OAuth Client and the protected resource server, and on which the access token is provided to the protected resource server by the OAuth Client.

Another solution for this token binding is proposed in the "OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens" as defined in IETF RFC 8705 [23]. In this solution the access token is bound to the fingerprint of public key of the client, used during the mutual authentication between the client and the authorization server. The resource server in the same way obtains the public key from the TLS stack and verifies the fingerprint with the one contained in the access token before giving access to the protected resources to the client.

A.3.1 OAuth 2.0 Token Binding

A.3.1.1 Token Binding ID

The token binding for access tokens cryptographically binds the access token to the client's Token Binding key pair, possession of which is proven on the TLS connection between the client and the protected resource.

The Token Binding Protocol as defined in IETF RFC 8471 [i.3] defines Token Binding ID for a TLS connection between a client and a server. This Token Binding ID is constructed using the public key of a private-public key pair. The client proves the possession of the corresponding private key.

An attacker need to be able to use the Client's private key to export and replay a bound security token. To avoid this attack, the private key should be specially protected, e.g. generated in a Hardware Security Module.

The Token binding ID is then used in the "Sec-Token-Binding" header as the referred Token Binding ID and is used to Token bind the access token. The authorization server associates and embeds the Token Binding ID with the access token in a way that can be accessed by the protected resource server, for the access token verification.

NOTE: To obtain the Token Binding ID, the client may need to establish the TLS connection between itself and the protected resource server prior to making the token request to the authorization server.

A.3.1.2 Token Binding for ID Token

It is possible to add a Token Binding in the OpenID Connect ID Tokens. A new 'tbh' (token binding hash) element is defined in the confirmation claim "cnf" to represent the SHA-256 hash of a Token Binding ID in an ID token.

Table A.3.1.2-1: Token Binding for ID token

Claim	Method	Mandatory/ Optional/ Conditional	Description	Value
cnf	tbh	O	Token Binding ID hash value	The value of the "tbh" member is the base64url encoding of the SHA-256 hash of the Token Binding ID.

A.3.1.3 Advantage of Token Binding

The access token is bound to the TLS connection used between the OAuth Client and the protected resource server, and on which the access token is provided to the protected resource server by the OAuth Client. The Access Token cannot be used by an attacker on another TLS connection and this avoids the misuse of access token.

A.3.1.4 Security considerations

A.3.1.4.1 Security Token Replay

The Token Binding private keys are high-value assets and should be strongly protected ideally generating them in a hardware security module that prevents key export.

The bound token needs to be integrity-protected, so that an attacker cannot remove the binding or substitute a Token binding ID of their choice without detection.

But nothing prevent collaborative Client to export a bound token with corresponding Token Binding private key.

A.3.1.4.2 Downgrade attacks

The Token Binding protocol is negotiated using a mechanism that prevents downgrade, e.g. use TLS extension for Token Binding negotiation.

A.3.2 OAuth 2.0 Certificate Bound Access Tokens

A.3.2.0 Basic principle

OAuth 2.0 certificate bound Access Tokens is described in "OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens" as defined in IETF RFC 8705 [23].

When mutual TLS authentication is used by the client on the connection of the token endpoint, the authorization server is able to bind the access token to the client certificate. This binding is accessible by the protected resource server, either in the issued access token directly or through the Token Introspection process described in "OAuth 2.0 Token Introspection" as defined in IETF RFC 7662 [21].

This way of binding the access token to the client has the advantage to decouple the binding of the access token (from the client's authentication with the authorization server) and the proof-of-possession mechanism (enabled by the client's authentication with the protected resource server). The only constraint is the use of the same certificate for the client's authentication with the authorization server and for the client's authentication with the protected resource server.

A.3.2.1 Certificate bound access token using JWT

The certificate hash information is included in the confirmation method of the JWT ("cnf"). A new confirmation method member ("x5t#S256") has been defined to convey the certificate hash information.

Table A.3.2.1-1: Certificate hash element for access token

Claim	Method	Mandatory/ Optional/ Conditional	Description	Value
cnf	x5t#S256	O	X.509 Certificate SHA-256 Thumbprint hash value	The value of the " x5t#S256" member is the SHA-256 hash of the DER encoding of the X.509 certificate base64url-encoded with all trailing pad '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

Oher JWT confirmation method members could be defined in the future if needed.

The same confirmation method is applicable for the introspected access token and is included in the token introspection response.

A.3.3 OAuth 2.0 Token Binding and OAuth2.0 Certificate Token binding comparison

In OAuth 2.0 Token Binding, the key material is automatically managed by the TLS stack. In the OAuth2.0 Certificate Token binding, the developer creates and maintains the key pairs and respective certificates. The use of self-signed certificates facilitates and reduces the complexity of this solution.

OAuth 2.0 Token Binding allows to use different key pairs for different resource servers, which is a privacy benefit. In the case of NFV, the privacy should not be an issue.

But OAuth 2.0 Token Binding needs the use of TLS extensions for the token binding that are not largely used for the time of the present document writing. The OAuth2.0 Certificate Token binding requires only widely deployed TLS features and then easier to adopt in a short term.

A.4 3GPP authorization framework

A.4.0 OAuth 2.0 authorization in 3GPP

The 3GPP has defined in ETSI TS 133 501 [i.6] an authorization framework for the authorization of Network Functions service access for 5G systems. This authorization framework uses the OAuth 2.0 framework specified in IETF RFC 6749 [6], and its support by Network Functions (NF) and Network Resource Function (NRF), acting as OAuth 2.0 Authorization Server, is mandated by 3GPP.

The Grant type used in 3GPP is the Client Credentials Grant, as the grant type defined by ETSI GS NFV-SOL 013 [22].

Access token are JSON Web Tokens and secured with digital signatures or Message Authentication Code (MAC) as defined in JSON Web Signature (JWS) [10].

A.4.1 Authentication between Network Functions

When the token-based authorization is used 3GPP mandates that the service consumer NF authenticates the Service Producer NF at transport layer before trying to access to the service API. 3GPP allows the Service producer NF to authenticate the service consumer NF; the authentication of the Service consumer NF is implicit using the token-based authorization, which is granted only after a mutual authentication of the service consumer NF towards the NRF at transport layer.

A.4.2 Access Token Request

Before the Network Function that consumes a service (OAuth 2.0 Client), is able to request an access token, this Network Function first registers with the NRF acting as the Authorization Server using the NF service registration procedure and using the Client id which is the NF Instance Id of the NF.

In the Access Token Request the NF consumer includes its NF Instance Id and its NF type, the expected NF service name and NF Type. These information allows the NRF to verify if the NF consumer is authorized to access to this NF producer and NF service. If the NF service consumer is authorized, an access token with appropriate claims is generated and sent back to the NF service consumer.

A.4.3 3GPP Access Token

3GPP mandates that the 3GPP access token is JSON Web Token as described in IETF RFC 7519 [8].

The 3GPP access token is secured with digital signatures or MAC based on JSON Web Signature (JWS) as described in IETF RFC 7515 [10].

3GPP mandates that the claims in the JSON Web Token include:

- NF Instance Id of the NRF (issuer of the access token).
- NF Instance Id of the NF Service consumer (subject).
- NF Instance Id of the NF Service producer (audience).
- Authorized services (scope).
- Expiration time (expiration).

Additional claims may be further defined by 3GPP.

These claims ensure that the access token is bound to the issuer of the access token, bound to the subject and cannot be used by another malicious NF, and bound to the service producer and cannot be used for another service.

A.4.4 Service access request

The service consumer NF request access to the service provided by the service producer NF including the access token in the request and after a successful authentication towards the service producer NF.

The service producer NF verifies the access token or sends it to the NRF for verification. The verification consists of integrity check and verification of the claims in the token. If the service producer NF verifies the access token by itself, it needs the NRF's public key or the shared secret that has been used by the NRF to generate the access token.

Annex B (informative): Synthesis on existing Access Token

Table B-1 compares the existing solutions described in annex A against the security requirements defined in clause 4.3.

NOTE 1: The text reproduced in table B-1 was extracted from clause 4.3 of the present document and from other external documents for readability purposes. Requirements reproduced in this table are to be considered as quotes: they are not new requirements.
In the unlikely case (following re-publication of any of the quoted documents) where a description or a comment in table B-1 would differ from the source text, it is the source text that takes precedent.

NOTE 2: The empty cells mean that information found on the existing access token technology is not sufficient to assess the fulfilment of the corresponding requirement.

Table B-1: Synthesis on existing Access Token

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Auth-Prot_001	The confidentiality of the requests shall be ensured by using a transport-layer mechanism such as TLS on each interface.			X	Solution provided in IETF RFC 6819 [i.4] + OpenID connect provides a way to provide confidentiality of the request: content of the request is an encrypted JWT.	X	A protection at transport layer is used (e.g. TLS).	X	Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23], a TLS channel is established.
Auth-Prot_002	The client and authorization servers shall mutually authenticate.			X	IETF RFC 6819 [i.4] + authentication of the server through either the use of signed or encrypted JWT with appropriate key and cipher.	X	Mutual authentication is done by the transport layer protection and is required.	X	Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23].

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Auth-Prot_003	The client shall authenticate the resource server.					X	The service Consumer NF shall authenticate the service producer NF.	X	Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23], a mutual authentication is required between the client and protected resource server. Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the Client authenticates to the protected resource server and the public key of the client is used to generate the token binding ID.
Auth-Prot_004	Before accepting the token as valid, the resource server shall authenticate the originator of the request as the legitimate owner of the token.			X	The token is bound to the subject through the subject Identifier, ensuring that the token has been provided for this consumer. In OpenID connect the connection to the resource server is not described.	X	Authentication of the service consumer NF towards the service producer NF will be implicit by authorization, which can only be granted after successful authentication of the service consumer NF towards the NRF.	X	Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23], the protected resource server shall verify that the certificate used for the mutual authentication is the same as the certificate associated to the access token. Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the protected resource server verifies that the token binding ID in access token is the correct one.

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Auth-Prot_005	The Authorization server database used to authenticate the client and store associated client credentials, access tokens and refresh tokens shall be stored in a tamper resistant location (e.g. HSM).				This is an implementation requirement that is not described in the specification.		Depends on implementation, not specified by 3GPP.		Depends on implementation of authorization server.
Client-Cred_001	The client credentials shall be stored in a secure and tamper resistant location or stored encrypted with a key protected in a tamper resistant location.						Depends on implementation, not specified by 3GPP.		
Client-Cred_002	The client credentials shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources, in order to mitigate the risk of guessing attacks.								
Client-Cred_003	The client credentials shall not be included in the source code and software packages.								

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Client-Cred_004	The client credentials shall be installed in the client in a secure way eliminating any possibility of gaining access to these credentials during installation.								
Client-Cred_005	It shall be possible for the authorization server to revoke the client credentials.								
Acc-Token_001	The access token shall be stored in a secure and tamper resistant location or stored encrypted with a key protected in a tamper resistant location.		Depends on the implementation of Keystone fernet key repository and the protection of the process of encryption decryption of the fernet token.				Depends on implementation, not specified by 3GPP.		
Acc-Token_002	The access token shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources, in order to mitigate the risk of guessing attacks.								
Acc-Token_003	Access tokens shall have policy-defined limited scope.	X	The token may be scoped token (Project, Domain or System scoped). For the Project scoped token, Project_ID is included in the token.	X		X	The access token includes a claim for the authorized services (scope).		
Acc-Token_004	Access tokens shall have limited lifetimes.	X	The token includes an expiration time and a timestamp.	X	Lifetime and timestamp values may be included in the token.	X	The access token includes a claim for the expiration time (expiration).		

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Acc-Token_005	Access tokens shall be restricted to a particular number of operations.	X	The Fernet Token are ephemeral bearer tokens. They are encrypted with a key that the system may rotate. Tokens encrypted with a primary key can be decrypted, and validated, on other nodes where that key is still staged.	X	A nonce value used to associate a Client session with an ID Token, and to mitigate replay attacks could be added in the "nonce" claim.			X	Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7]: The access token is bound to a key material (token binding Id, which is associated to the TLS connection between the client and the resource server. This solution is a way to mitigate the replay attacks. Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23]: The access token is associated to the sender via the fingerprint of its public key. This is a way of mitigation of replay attacks by a malicious client.
Acc-Token_006	It shall be possible to bind the access token to the intended resource server.		Not really. The token could just be a Project/Domain or System scoped token.	X	With the "aud" claim.	X	The access token includes a claim for the NF Instance Id of the Service Producer (audience).	X	Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the token binding ID may be specific to the resource server implied in the TLS connection.
Acc-Token_007	It shall be possible to bind the token to the endpoint URL (token audience) used to obtain the token.				Claims (e.g. "aud") to bind to the Id (client Id) but not the URL.			X	Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the token is bound to the TLS connection.
Acc-Token_008	It shall be possible to limit the scope of the token and associate it to particular resource.		The token may be scoped token (Project, Domain or System scoped). For the Project scoped token, Project_ID is included in the token. Not really precise to be able to associate to a particular resource.	X	With the "aud" claim.	X	The access token includes a claim for the authorized services (scope).		

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Acc-Token_009	Tokens shall be bound to the client ID.	X	The Fernet token includes the User ID.	X	The ID token associated to access token (bound with the "at_hash" claim in ID token) has "aud" claim used to bind the access token to the "client_id".	X	The access token includes a claim for the NF Instance Id of the Service Consumer (subject) which is the "Client ID".	X	Using the IETF RFC 8705: "OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens" [23], the access token is bound to the certificate of the client. Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the token is bound to the public key used for the TLS connection with the resource server.
Acc-Token_010	The access token shall be signed to detect manipulation of the token or production of fake tokens.	X	Signed using SHA256 HMAC (with a 128 bits key).	X	The token shall be signed using JWS.	X	Access tokens are secured with digital signatures or Message Authentication Codes (MACs) based on JSON Web Signature (JWS) as described in [10].	X	Using IETF RFC 7515: "JSON Web Signature (JWS)" [10]. Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], a solution for token binding using JWT is described. The use of JWS is then possible but not described.
Acc-Token_011	It shall be possible to encrypt content of the access token.	X	Encrypted with AES128 in CBC mode using an Initialization vector IV included in the token.	X	Use of JWE is possible in addition of JWS.		The use of JWE is not described.	X	IETF RFC 7516: "JSON Web Encryption (JWE)" [11].
Acc-Token_012	The access token should be defined in a standard format (SAML or JWT).		No Fernet token format is not standard and the fernet spec is abandoned. Openstack is working on the addition of JWT, JWS and JWE to rely on standard format.	X	ID token is represented as JSON Web Token (JWT).	X	Access tokens shall be JSON Web Tokens as described in [8].	X	IETF RFC 7519: "JSON Web Token (JWT)" [8]. Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], a solution for token binding using JWT is described.
Acc-Token_013	It shall be possible to revoke an access token.	X	Rotation of the key. The fernet token is really revoked when all nodes have rotated the keys and the key used for encryption of the fernet token is no more available.	X	The authorization server should provide a mechanism for this revocation. If not the lifetime of the Access token shall be very short or access token should be single use.		Not described.		

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Ref-Token_001	The refresh token shall be stored in a secure and tamper resistant location or stored encrypted with the key protected in a tamper resistant location.		NA		Depends on implementation.				
Ref-Token_002	The refresh token shall be generated with a minimum of 128 bits of entropy, using best practices for entropy sources [12], in order to mitigate the risk of guessing attacks.		NA	X					
Ref-Token_003	Refresh tokens shall have policy-defined limited scope.		NA						
Ref-Token_004	Refresh tokens shall have limited lifetimes.		NA						
Ref-Token_005	Refresh tokens shall be restricted to a particular number of operations.		NA						
Ref-Token_006	The refresh token shall be bound to the client ID.		NA					X	Using draft-ietf-oauth-token-binding: "OAuth 2.0 Token Binding" [i.7], the token is bound to the public key used for the TLS connection with the resource server.

Requirements in clause 4.3		Openstack Keystone		OpenId Connect		ETSI TS 133 501 [i.6]		IETF	
Number	**Description**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**	**Req OK**	**Comments**
Ref-Token_007	It shall be possible to rotate refresh tokens by changing the value of the refresh token with every refresh request.		NA						
Ref-Token_008	It shall be possible to revoke a refresh token.		NA						

Annex C (informative): IANA Registry Considerations

C.1 "Well-Known URIs" Registry

C.1.1 Introduction

In order to allow the discovery of the authorization server configuration, as described in clause 5.1.4 of the present document, some information is registered in the IANA "Well-Known URIs" registry.

Clause C.1.2 describes the content of the registered information.

NOTE: Well-known URIs registration is described in IETF RFC 8615 [17].

C.1.2 Registry contents

- URI suffix: `nfv-oauth-server-configuration`
- Change controller: ETSI: PNNS@etsi.org
- Specification document: clause 5.1.4 of the present document
- Related information: None

C.2 JSON Web Token Claims registry

C.2.1 Introduction

Several Claims described in clause 5.5 of the present document are declared in the IANA JSON Web Token registry.

Clause C.2.2 describes the content of the registered information.

NOTE: JSON web token are specified in IETF RFC 7519 [8].

C.2.2 Registry contents

- Claim Name: `at_use_nbr`
- Claim Description: Number of API requests for which the access token can be used
- Change Controller: ETSI: PNNS@etsi.org
- Specification Document(s): clause 5.5 of the present document

C.3 OAuth Parameters registry

C.3.1 Introduction

Some OAuth parameters associated with the access token are declared in the IANA OAuth Parameters registry.

Clause C.3.2 describes the content of the registered information.

NOTE: OAuth parameters are specified in IETF RFC 6749 [6].

C.3.2 Registry contents

- Parameter name: `nfv_token`
- Parameter usage location: Access Token Response
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.4 of the present document
- Related information: None

C.4 OAuth Dynamic Client Registration Metadata registry

C.4.1 Introduction

Some OAuth Dynamic Client Registration metadata is declared in the IANA OAuth Dynamic Client Registration Metadata registry.

Clause C.4.2 describes the content of the registered information.

NOTE: OAuth Dynamic Client Registration Metadata are specified in IETF RFC 7591 [18].

C.4.2 Registry contents

- Client Metadata Name: `nfv_token_signed_response_alg`
- Client Metadata Description: JWS alg algorithm required for signing the nfv Token issued to this Client
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.2.3 of the present document
- Related information: None
- Client Metadata Name: `nfv_token_encrypted_response_alg`
- Client Metadata Description: JWE alg algorithm required for encrypting the nfv Token issued to this Client
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.2.3 of the present document
- Related information: None
- Client Metadata Name: `nfv_token_encrypted_response_enc`
- Client Metadata Description: JWE enc algorithm required for encrypting the nfv Token issued to this Client
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.2.3 of the present document
- Related information: None

C.5 OAuth Authorization Server Metadata registry

C.5.1 Introduction

Some OAuth Authorization Server Metadata is declared for the OAuth Authorization Server configuration in the Authorization Server Metadata registry.

Clause C.5.2 describes the content of the registered information.

NOTE: OAuth Authorization Server Metadata are specified in IETF RFC 8414 [13].

C.5.2 Registry contents

- Client Metadata Name: `nfv_token_signing_alg_values_supported`
- Client Metadata Description: JSON array containing a list of the JWS signing algorithms supported by the server for signing the JWT used as NFV Token
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.1.4 of the present document
- Related information: None
- Client Metadata Name: `nfv_token_encryption_alg_values_supported`
- Client Metadata Description: JSON array containing a list of the JWE encryption algorithms (alg values) supported by the server to encode the JWT used as NFV Token
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.1.4 of the present document
- Related information: None
- Client Metadata Name: `nfv_token_encryption_enc_values_supported`
- Client Metadata Description: JSON array containing a list of the JWE encryption algorithms (enc values) supported by the server to encode the JWT used as NFV Token
- Change controller: ETSI: PNNS@etsi.org
- Specification document(s): clause 5.1.4 of the present document
- Related information: None

Annex D (informative): Change history

Date	Version	Information about changes
2018-01-18	V0.0.1	First draft with the Table of Content.
2018-01-19	V0.0.2	Implementation of the following contributions accepted during the SEC#117 meeting: - NFVSEC(18)000004r1_SEC022_Introduction-draft - NFVSEC(18)000005r1_SEC022_Scope
2018-07-03	V0.0.3	Implementation of the following contributions accepted during the SEC#126 meeting: - NFVSEC(18)000052r1_SEC022_Section_5_1 - NFVSEC(18)000053r1_SEC022_Section_5_2
2018-07-06	V0.0.4	Implementation of the following contributions accepted during the SEC#127 meeting: - NFVSEC(18)000051r2_SEC022_Section_4_1 - NFVSEC(18)000054r1_SEC022_Section_5_3
2018-07-19	V0.0.5	Implementation of the following contributions accepted during the SEC#128 meeting: - NFVSEC(18)000073r1_SEC022_Section_4_2 with an editorial change for a.4.1.8: "open redirector on client" change to "redirection on client to malicious server" - NFVSEC(18)000074r1_SEC022_Section_4_3
2018-09-07	V0.0.6	Implementation of the following contributions accepted during the SEC#130 meeting - NFVSEC(18)000087r1_SEC022_Annex_A_3GPP_Token - NFVSEC(18)000088_SEC022_Annex_A_IETF_Mutual_TLS - NFVSEC(18)000089r1_SEC022_Section_5
2018-10-26	V0.0.7	Implementation of the following contributions accepted during the SEC#133 meeting - NFVSEC(18)000107r1_022_Requirements_cleanup reversing the Auth-Prot_003 change, as described in the report of the SEC#133 (NFVSEC(18)000128)
2018-11-29	V0.0.8	Implementation of the following contributions accepted during the SEC#135 meeting - NFVSEC(18)135001_SEC022_Authorization_Server_discovery adding a note: "NOTE: It is FFS how we establish trust with the Authorization Server for dynamic discovery". As described in SEC#135 meeting report (NFVSEC(18)000140) - NFVSEC(18)135003_SEC022_Registration_process
2018-12-05	V0.0.9	Implementation of the following contributions accepted during the SEC#136 meeting - NFVSEC(18)135007r1_SEC022_Access_Token_Format_and_metadata - NFVSEC(18)000152_SEC022_Token_request with Editorial corrections - NFVSEC(18)000153_SEC022_Token_verification_process
2019-01-22	V0.0.10	Implementation of Editorial comments following the SOL review Adding a clause for IANA registration
2019-02-13	V0.0.11	Editorial modifications after EditHelp. - Hanging paragraph suppression - Changes in the IANA Registry consideration Clause 7 - Add IANA registration that was missing. - Editorial changes to make the NFV Token naming consistent - Change the cardinality of optional elements to be consistent with SOL rules. - Change the scope to apply to all API of NFV-MANO endpoints, addressed by SOL013.
2019-02-20	V0.0.12	Editorial modifications and answers to the comments of NFV(19)000050: SEC022 Comments and editorial updates - Authorization Server support of the MTLS mandatory in Authorization server configuration. - Add example of protected resource request with access_token and nfv token.
2019-04-11	V0.0.13	Implementation of the following contributions accepted during the SEC#142, SEC#144 and SEC#145 meetings: - NFVSEC(19)000028_SEC022_-_Clause_5_2_-_Simplification_of_the_resgitration_pro - NFVSEC(19)000044r1_SEC022_Authorization_Server_Identifier_clarification - NFVSEC(19)000045r1_SEC022_single_access_token - NFVSEC(19)000047r1_SEC022_Authorization_Server_Configuration_simplification - NFVSEC(19)000049r1_SEC022_at_use_nbr_default_fix
2019-04-23	V0.1.0	Implementation of the following contribution accepted during the SEC#146 meeting with other changes agreed during the meeting: - NFVSEC(19)000059_DCM_comments_on_SEC022_v0_0_13
2019-08-06	V2.6.1	Publication
2019-11-21	V2.6.2	Implementation of the CR NFVSEC(19)000094r1 agreed during NFVSEC#152-F2F Paris
2020-01	V2.7.1	Publication
2020-04-09	V2.7.2	Implementation of the CR NFVSEC(20)000021 agreed during NFVSEC#161

Date	Version	Information about changes
2021-12-20	V3.5.1	New baseline for ETSI GS NFV-SEC 022 for release 3 (ed 3.6.1). Copy of the last published version 2.8.1
2023-10-25	V4.4.1	Updated for release 4 and addition of JSON Web Signature Tokens from contribution NFVSEC(23)000219r1.

History

Document history		
V4.5.1	January 2024	Publication