token-generation HTTP/TLS/JSON

Interface Design Description

Abstract
This document describes a HTTP protocol with TLS payload security and JSON payload encoding variant of the **token-generation** service.

Contents

1 Overview

This document describes the **token-generation** service interface, which enables session control during a service consumption. It's implemented using protocol, encoding as stated in the following table:

Profile type	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	TLS	1.3
Encoding	JSON	RFC 8259 [1]
Compression	N/A	-

Table 1: Communication and semantics details used for the **token-generation** service interface

This document provides the Interface Design Description IDD to the *token-generation – Service Description* document. For further details about how this service is meant to be used, please consult that document.

The rest of this document describes how to realize the *token-generation* service HTTP/TLS/JSON interface in details.

2 Interface Description

The service responses with the status code `200 Ok` if called successfully. The error codes are, `400 Bad Request` if request is malformed, `401 Unauthorized` if improper client side certificate is provided, `500 Internal Server Error` if Authorizator is unavailable.

```
POST /authorization/token HTTP/1.1

{
  "consumer": {
    "address": "string",
    "authenticationInfo": "string",
    "metadata": {
      "additionalProp1": "string",
      "additionalProp2": "string",
      "additionalProp3": "string"
    },
    "port": 0,
    "systemName": "string"
  },
  "consumerCloud": {
    "name": "string",
    "operator": "string"
  },
  "providers": [
    {
      "provider": {
        "address": "string",
        "authenticationInfo": "string",
        "metadata": {
          "additionalProp1": "string",
          "additionalProp2": "string",
          "additionalProp3": "string"
        },
        "port": 0,
        "systemName": "string"
      },
      "serviceInterfaces": [
        "string"
      ],
      "tokenDuration": 0
    }
  ],
  "service": "string"
}
```

Listing 1: An token-generation invocation.

```
{
  "tokenData": [
    {
      "providerAddress": "string",
      "providerName": "string",
      "providerPort": 0,
      "tokens": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
      }
    }
  ]
}
```

Listing 2: An token-generation response.

3 Data Models

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.9, which are used to represent things like hashes, identifiers and texts.

3.1 struct TokenGenerationRequest

Field	Type	Mandatory	Description
consumer	SystemDescriptor	yes	Descriptor of the consumer system.
consumerCloud	CloudDescriptor	no	Descriptor of the consumer cloud.
providers	List<TokenGenerationDescriptor>	yes	Array of token generation descriptors
service	Name	yes	Identifier of the service.

3.2 struct CloudDescriptor

Field	Type	Mandatory	Description
name	Name	yes	Name of the cloud
operator	Name	yes	Name of the cloud operator

3.3 struct SystemDescriptor

Field	Type	Mandatory	Description
address	Address	yes	Network address.
authenticationInfo	String	no	Public key of the client certificate.
metadata	Metadata	no	Metadata
port	PortNumber	yes	Port of the system.
systemName	Name	yes	Name of the system.

3.4 struct TokenGenerationDescriptor

Field	Type	Mandatory	Description
provider	SystemDescriptor	yes	Descriptor of the provider system.
interfaces	List<Name>	yes	List of interface names.
tokenDuration	Number	yes	Validity period of the token in seconds

3.5 struct TokenGenerationResponse

Field	Type	Description
tokenData	List<TokenData>	List of token data descriptors.

3.6 struct TokenData

Field	Type	Description
providerAddress	Address	Network address of the system.
providerPort	PortNumber	Port of the system.
providerName	Name	Name of the system.
tokens	Map<Interface,JWT>	Interface-token pairs

3.7 struct JWT

A String format credential (token) which were created by encrypting a JSON object [2]. JSON Web Tokens (JWT) are an open, industry standard RFC 7519 method for representing claims securely between two parties.

A JWT token genereated by this service, holds the following JSON object:

```
{
  "header": {
    "alg": "RS512",
    "typ": "JWT"
  },
  "payload": {
    "iss": "Authorization",
    "iat": 1516239022,
    "nbf": 1516239122, // optional
    "exp": 1516242622,
    "cid": "<system-name>.<cloud-name>.<cloud-operator>",
    "sid": "<service-name>",
    "iid": "<interface-identifier>",
  },
  "signature": "SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
}
```

Listing 3: Content of a JWT token generated by this service.

- The **alg** field specifies the cryptographic algorithm used to sign the JWT. Value is always `RS512`.
- The **typ** field is used to declare the media type of the JWT. Value is always `JWT`.
- The **iss** is used to identify the entity that issued the JWT. Value is always `Authorization`.
- The **iat** stands for ”issued at”, which represents the Unix timestamp when the token was created.
- The **nbf** stands for ”not before”, which represents the Unix timestamp before which the token should not be considered as valid.
- The **exp** stands for ”expiration time”, which represents the Unix timestamp after which the token is no longer valid.
- The **cid** stands for ”consumer identifier” and used to identify the consumer system which the token was generated for. The format of the value is always `<system-name>.<cloud-name>.<cloud-operator>`.
- The **sid** stands for ”service identifier” and used to identify the service which the token was generated for. The value is always the exact service definition name.
- The **iid** stands for ”interface identifier” and used to identify the interface which the token was generated for. The value is always the exact interface name.
- The **signature** is always coming from the private key of the issuer (Authorization Core System) and can be verified with the public key of the issuer.

3.8 struct Metadata

An Object which maps String key-value pairs.

3.9 Primitives

As all messages are encoded using the JSON format [2], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [3], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the **orchestration-service** SD document without a direct equivalent among the JSON types. Concretely, we define the **orchestration-service** SD primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.9.1 alias Address = String

A string representation of a network address. An address can be a version 4 IP address (RFC 791), a version 6 IP address (RFC 2460) or a DNS name (RFC 1034).

3.9.2 alias Interface = String

A String that describes an interface in *Protocol-SecurityType-MimeType* format. *SecurityType* can be SECURE or INSECURE. *Protocol* and *MimeType* can be anything. An example of a valid interface is: ”HTTP-SECURE-JSON” or ”HTTP-INSECURE-SENML”.

3.9.3 alias List <A> = Array<A>

There is no difference.

3.9.4 alias Name = String

A String identifier that is intended to be both human and machine-readable.

3.9.5 alias PortNumber = Number

Decimal Number in the range of 0-65535.

4 References

[1] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, Dec. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8259.txt

[2] ——, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[3] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	YYYY-MM-DD	4.6.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	4.6.0	Xxx Yyy