

```
POST /oauth2/revoke HTTP/1.1
Host: www.domain.org
Content-Type: application/x-www-form-urlencoded

token=_TiHRG-bA-H3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw&
token_type_hint=refresh_token&
client_id=<OAuth2_client_id>&
client_secret=<OAuth2_client_secret>&
clientId=12345678
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/x-www-form-urlencoded"
-d 'token=_TiHRG-bA-H3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw&
  token_type_hint=refresh_token&
  client_id=<OAuth2_client_id>&
  client_secret=<OAuth2_client_secret>&
  clientId=12345678'
https://www.domain.org/oauth2/revoke
```

Sample Response

```
HTTP/1.1 204 No Content
```

8.5 Authentication and authorization for electronic seals

8.5.1 Introduction

The eIDAS regulation (Regulation (EU) No 910/2014 [i.1]) defines two basic concepts: an electronic signature, created by a natural person used to sign the content of a document, and an electronic seal based on a certificate of a legal person used to prove the origin and integrity of the document. From a mere technical point of view, both electronic signatures and electronic seals are digital signatures. However, the usage of the CSC API in order to create an electronic signature or an electronic seal can be different depending on the above-cited legal context. The present section discusses the usage of the CSC API for creating electronic seals, which in the context of the present document are digital signatures created by using a certificate issued to a legal person. This ensures the integrity and origin of the document, without necessarily committing to the content. ::: {.NOTE} This definition is not limited to the legal definition of electronic seals in Regulation (EU) No 910/2014 [i.1]. ::: In many cases, electronic seals are created in automated processes and often a large number of documents are to be sealed in one session. In the present document, there are two different possible authorizations. The first one is the authorization to get access to the API, and the second one is the authorization to use the signing credential for the seal/signature creation. The following section describe how these authorizations can be done with the purpose of creating an electronic seal.

8.5.2 Service authorization and authentication for electronic seals

Several methods allow access to the CSC API without the need for regular human interaction, which would not be very practical in the case of sealing a large number of documents.

8.5.2.1 Login / password

HTTP basic or HTTP digest authentication can be used to provide access to the API. The login and password MAY be linked to the signature application or to the certificate owner.

8.5.2.2 OAuth with client credentials grant

The usage of OAuth 2.0 with client credential grant allows granting access to the signing application. It does convey any user specific identifier. This authenticates the client, any user specific information is indicated within the respective CSC API call or provided implicitly or separately.

8.5.2.3 Mutual TLS

The signing server can be configured to use TLS connections, requiring clients that attempt to connect to get authenticated. A client SHALL use a client certificate in order to authenticate. The client certificate SHALL contain information allowing the signing server to authenticate the client application/user. The signing server MAY be configured to accept TLS connections only from a limited group of allowed clients.

An example can just be a scenario where the usage of the sealing credentials is limited only to successfully authenticated TLS connections using client certificates authentication connections. This method does not create any token. In an additional use case, the remote signing service provider has a specific end point (outside of CSC specification) which can be accessed via TLS authentication + API key + secret which creates an Access Token. And this access token is used later on to access the API. In case of seals, no extra authorization is used to access the private key. Used with short lived credentials. ::: {NOTE} By defining an empty set of authentication object types, the RSSP can decide to not need any more actions. :::

In addition to the mutual TLS, a token can be created based on login / password + OTP by a non CSC end point, and is then used for signing together with a PIN. This can be used with long-term certificates

8.5.3 Credential authorization for electronic seals

The credential authorization allows the usage of a specific key. There are three possible strategies, to avoid human interaction for each signature.

- The first is the usage of an authorization means that can be fully automatized, for example the usage of a PIN.
- It is also possible to not require any additional actions, if the access token is already sufficient.
- The third one, consists in creating a SAD for a high but limited number of signatures. Since the creation of the SAD is an operation which is not repeated very often, it can be created in a non-fully automated process. This allows a more complex authorization, and to be more precise in what this authorization includes.

9 Creating a remote signature

Remote signature services allow generating digital signatures remotely by means of an RSCD operated as a service. An RSSP is an organization that manages the RSCD on behalf of the signers.

In general, each time a remote signature is required, a strong authentication mechanism SHOULD be invoked. Strong authentication requiring the user to authorize to the signature application multiple times in a rapid sequence using authorization mechanisms like OTP can be cumbersome. In order to improve the signer's experience, the strong authentication MAY be allowed to occur only once per signing session (for example with a single OTP) covering multiple signatures.

The current specification supports the following three use cases:

1. The remote signature of a single hash;
2. The remote signature of multiple hashes passed in a single signature operation;
3. The remote signature of multiple hashes passed across multiple signature operations occurring within a single signing session.

A RSSP SHALL support at least case 1, with credentials authorization occurring every time a signature is created.

The RSSP decides whether to support multi-signature transactions (use cases 2 and 3) or not. In some cases, regulatory or security requirements may forbid multi-signature transactions. The *multisign* output value of the **credentials/info** method, as defined in [credentials/info](#), provides information if multi-signature transactions are supported by a specific credential or not.

A multi-signature transaction can be created by invoking the **signatures/signHash** method, as defined in [signatures/signHash](#), and submitting multiple hash values in one run (use case 2, suitable for "batch signing" of multiple documents) or by invoking **signatures/signHash** multiple times (use case 3, suitable for creating multiple signatures from a single user in a PDF document). In both cases, the authorization mechanism adopted by the signature application SHALL explicitly specify the total number of signatures to be authorized and the remote signing service SHALL prevent signature applications from creating more signatures than authorized.

See [Interaction among elements and components](#) to understand the workflows supported in this specification and the sequence of API calls to be invoked to create the supported types of remote signatures.

10 Error handling

Errors are returned by the remote service using standard HTTP status code syntax. Additional information is included in the body of the response from an API request using JSON.

The HTTP protocol defines a list of standard status codes that are referenced in this specification to help the signature application deal with these responses accordingly. For the events described in Table 2, the remote service SHALL support the corresponding HTTP status codes.

Table 2 – Supported HTTP Status Codes

Standard Status Code	Description
200 OK	Response to a successful API method request.
204 No Content	Response to a successful API method request in case no content is returned.
302 Found	Response used to redirect the user to an OAuth 2.0 authorization endpoint.
400 Bad Request	Returned due to unsupported, invalid or missing required parameters.

Standard Status Code	Description
401 Unauthorized	Returned when a bad or expired authorization token is used.
429 Too Many Requests	Returned when a request is rejected due to rate limiting.
500 Internal Server Error	Returned when the server encounters an unexpected condition.
501 Not Implemented	Returned when an unimplemented method is requested.
503 Service Unavailable	Returned when the server is currently unable to handle the request due to temporary overloading or maintenance conditions.

Status codes 429 and 50x are applicable to the remote service overall and are not specific to any API methods. For this reason, they are not mentioned in the error tables for each method specifically.

10.1 Error messages

Just as an HTML error page shows a useful error message to a visitor, the remote service implementing the API described in this specification SHALL provide a useful error message in case something goes wrong. When an error is detected, the remote service SHALL return the corresponding HTTP status code and SHALL return the information on the error in the body of the HTTP response using the “application/json” media type, as defined by RFC 4627 [5]. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings as shown in the following example:

```
HTTP/1.1 400 Bad Request
Date: Mon, 03 Dec 2018 12:00:00 GMT
Content-Type: application/json; charset=utf-8
Content-Length: ...
{
  "error": "invalid_request",
  "error_description": "The access token is not valid"
}
```

The *error_description* parameter is OPTIONAL but highly RECOMMENDED to provide a human-readable text string containing additional information to assist the user in understanding the error that occurred.

The remote service can also define custom error messages by using messages that are not defined in this specification.

The following table contains definitions for errors that are common to more than one API methods. Therefore, they’re presented only once in this section instead of being repeated for all API methods.

Table 3 – Predefined common Error Messages

Error	Error Description
invalid_request	The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.
unauthorized_client	The client is not authorized to use this method.
access_denied	The user, authorization server or remote service denied the request.
unsupported_response_type	The authorization server does not support obtaining an authorization code using this method.

Error	Error Description
invalid_scope	The requested scope is invalid, unknown, or malformed.
server_error	The authorization server encountered an unexpected condition that prevented it from fulfilling the request.
temporarily_unavailable	The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
expired_token	The access or refresh token is expired or has been revoked.
invalid_token	The token provided is not a valid OAuth access or refresh token.

11 The remote service APIs

In order to simplify the navigation of this specification, the following table summarizes all the API methods defined in the present specification. The **info** method, as defined in [info](#), SHALL be implemented. All other methods are OPTIONAL.

Table 4 – API methods summary

API Method	Description
info	Returns information on the remote service and the list of API methods it has implemented.
auth/login	Authorize the remote service with HTTP Basic or Digest authentication.
auth/revoke	Revoke the service access token or refresh token.
credentials/list	Returns the list of credentials associated to a user.
credentials/info	Returns information on a signing credential, its associated certificate and a description of the supported authorization mechanism.
credentials/authorize	Authorize the access to the credential for signing.
credentials/extendTransaction	Extend the validity of a multi-signature transaction.
credentials/sendOTP	Start the online OTP mechanism associated to a credential.
signatures/signHash	Calculate a raw digital signature from one or more hash values.
signatures/signDoc	Creates one or more AdES signatures for documents or document digests.
signatures/timestamp	Return a time stamp token for the input hash value.
oauth2/authorize*	Initiate an OAuth 2.0 authorization flow.
oauth2/token*	Obtain an OAuth 2.0 access token or refresh token.
oauth2/revoke*	Revoke an OAuth 2.0 access token or refresh token.

Note 21: Although **oauth2/authorize** , **oauth2/token**, **oauth2/pushed_authorize**, and **oauth2/revoke**, as defined in [OAuth 2.0 Authorization](#), do not specify regular CSC API methods but rather endpoints managed by the OAuth2 authorization server, they're listed in Table 4 to provide a complete overview of the endpoints that can be supported by a remote service conforming to this specification.

11.1 info

Description

Returns information about the remote service and the list of the API methods it supports. This method SHALL be implemented by any remote service conforming to this specification.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
<i>lang</i>	OPTIONAL	<i>String</i>	Request a preferred language of the response to the remote service, specified according to RFC 5646 [9]. If present, the remote service SHALL provide language-specific responses using the specified language. If the specified language is not supported then it SHALL provide these responses in the language as specified in the <i>lang</i> output parameter.

11.1.0.1 Output:

This method returns the following values using the “application/json” format:

Attribute	Presence	Value	Description
<i>specs</i>	REQUIRED	<i>String</i>	The version of this specification implemented by the provider. The format of the string is Major.Minor.x.y , where Major is a number equivalent to the API version (e.g. 2 for API v2) and Minor is a number identifying the version update, while x and y are subversion numbers. The value corresponding to this specification is “2.0.0.0”.
<i>name</i>	REQUIRED	<i>String</i>	The commercial name of the remote service. The maximum size of the string is 255 characters.
<i>logo</i>	REQUIRED	<i>String</i>	The URI of the image file containing the logo of the remote service which SHALL be published online. The image SHALL be in either JPEG or PNG format and not larger than 256x256 pixels.
<i>region</i>	REQUIRED	<i>String</i>	The ISO 3166-1 [22] Alpha-2 code of the Country where the remote service provider is established (e.g. ES for Spain).
<i>lang</i>	REQUIRED	<i>String</i>	The language used in the responses, specified according to RFC 5646 [9].
<i>description</i>	REQUIRED	<i>String</i>	A free form description of the remote service in the <i>lang</i> language. The maximum size of the string is 255 characters.
<i>authType</i>	REQUIRED	<i>Array of String</i>	One or more values corresponding to the service authorization mechanisms supported by the remote service to authorize the access to the API: <ul style="list-style-type: none">• “external”: in case the authorization is managed externally (e.g. using a VPN or a private LAN).• “TLS”: in case the authorization is provided by means of TLS client certificate authentication.• “basic”: in case of HTTP Basic Authentication.• “digest”: in case of HTTP Digest Authentication.• “oauth2code”: in case of OAuth 2.0 with authorization code flow.• “oauth2client”: in case of OAuth 2.0 with client credentials flow.

Attribute	Presence	Value	Description
<i>oauth2</i>	REQUIRED Conditional	<i>String</i>	<p>The base URI of the OAuth 2.0 authorization server endpoint supported by the remote service for service authorization and/or credential authorization. The parameter SHALL be present if</p> <ul style="list-style-type: none"> the <i>authType</i> parameter contains “oauth2code” or “oauth2client” or the remote service supports the value “oauth2code” for the <i>auth/mode</i> parameter returned by credentials/info (as specified in credentials/info) <p>and the parameter “oauth2issuer” is not present.</p> <p>This URI SHALL be combined with the path components described in OAuth 2.0 Authorization in order to build the actual endpoint URLs.</p>
<i>oauth2issuer</i>	REQUIRED Conditional	<i>String</i>	<p>The issuer URL of the OAuth 2.0 authorization server as defined in IETF RFC 8414 [23] supported by the remote service for service authorization and/or credential authorization. The parameter SHALL be present if</p> <ul style="list-style-type: none"> the <i>authType</i> parameter contains “oauth2code” or “oauth2client” or the remote service supports the value “oauth2code” for the <i>auth/mode</i> parameter returned by credentials/info (as specified in credentials/info) <p>and the parameter “oauth2” is not present.</p> <p>The OAuth endpoint URLs are obtained from the OAuth Server metadata as described in IETF RFC 8414 [23].</p>
<i>asynchronousOperationMode</i>	OPTIONAL	<i>Boolean</i>	<p>This parameter shall be “true” if the remote signing server supports also asynchronous signature mechanism. The default value is “false”. An omitted parameter or the value “false” indicates that the remote signing server manages signature requests only in synchronous operation mode.</p>
<i>methods</i>	REQUIRED	<i>Array of String</i>	<p>The list of names of all the API methods described in this specification that are implemented and supported by the remote service.</p>
<i>validationInfo</i>	OPTIONAL	<i>Boolean</i>	<p>This parameter SHALL be “true” if the remote signing server supports the “validationInfo” response parameter of the method signatures/signDoc in not mandatory cases. An omitted parameter or the value “false” indicates that the remote signing server does not support “validationInfo” in those cases.</p>
<i>signAlgorithms</i>	REQUIRED	<i>JSON Object</i>	<p>Object including one or more signature algorithms supported by the RSSP.</p>
<i>signature_formats</i>	REQUIRED	<i>JSON Object</i>	<p>Object including one or more signature formats supported by the RSSP.</p>
<i>conformance_levels</i>	REQUIRED	<i>Array of String</i>	<p>The list of names of all signature conformance levels supported by the RSSP as defined in the Input parameter table in signatures/signDoc.</p>

The *signAlgorithms* is a JSON Object composed by the following parameters:

- *algos*
- *algoParams*

specified according to the following table.

Parameter	Presence	Value	Description
<i>algos</i>	REQUIRED	Array of String	The list of signature algorithms supported by the RSSP as defined in the Input parameter table in signatures/signHash . The supported signature algorithms SHOULD follow the recommendations of ETSI TS 119 312 [21] and SHALL be expressed as defined in Expressing algorithms clause.
<i>algoParams</i>	REQUIRED Conditional	Array of String	The list of eventual signature parameters as defined in the Input parameter table in signatures/signHash .

The *signature_formats* is a JSON Object composed by the following parameters:

- *formats*
- *envelope_properties*

specified according to the following table.

Parameter	Presence	Value	Description
<i>formats</i>	REQUIRED	Array of String	The list of signature formats supported by the RSSP as defined in the Input parameter table in signatures/signDoc .
<i>envelope_properties</i>	REQUIRED Conditional	Array of Array of String	The list of the properties concerning the signed envelope, whose possible values depend on the value of the <i>formats</i> parameter entries, as defined in the Input parameter table in signatures/signDoc . The number of arrays included in the <i>envelope_properties</i> array SHALL equal the number of entries in the <i>formats</i> array. The values included in the array at position <i>i</i> of the <i>envelope_properties</i> array SHALL refer to the signature format value included at position <i>i</i> of the <i>formats</i> array. An empty array at the position <i>i</i> of the <i>envelope_properties</i> array indicates that the RSSP supports the default signed envelope property for the signature format specified at the position <i>i</i> of the <i>formats</i> array, as defined in the Input parameter table in signatures/signDoc .

Note 22: *info* is a mandatory API method, so it MAY be excluded from the list of API method names returned by the *methods* parameter.

The endpoints **oauth2/authorize**, **oauth2/token**, **oauth2/pushed_authorize** and **oauth2/revoke**, as defined in [OAuth 2.0 Authorization](#), do not specify regular API methods but rather endpoints managed by the OAuth2 authorization server, therefore they MAY be excluded from the list of API method names returned by the *methods* parameter.

Sample Request

```
POST /csc/v2/info HTTP/1.1
Host: service.domain.org
Content-Type: application/json

{}
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/json"
-d '{}'
https://service.domain.org/csc/v2/info
```

Sample Response

```

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "specs": "2.0.0.0",
  "name": "ACME Trust Services",
  "logo": "https://service.domain.org/images/logo.png",
  "region": "IT",
  "lang": "en-US",
  "description": "An efficient remote signature service",
  "authType": ["basic", "oauth2code"],
  "oauth2": "https://www.domain.org/",
  "methods": ["auth/login", "auth/revoke", "credentials/list",
    "credentials/info", "credentials/authorize",
    "credentials/sendOTP",
    "signatures/signHash"],
  "signAlgorithms":
  {
    "algos": ["1.2.840.10045.4.3.2", "1.2.840.113549.1.1.1", "1.2.840.113549.1.1.10"]
  },
  "signature_formats":
  {
    "formats": ["C", "X", "P"],
    "envelope_properties": [
      ["Detached", "Attached", "Parallel"],
      ["Enveloped", "Enveloping", "Detached"],
      ["Certification", "Revision"]
    ]
  },
  "conformance_levels": ["Ades-B-B", "Ades-B-T"]
}

```

11.2 auth/login

Description

Obtain an access token for service authorization from the remote service using HTTP Basic Authentication or HTTP Digest authentication, as defined in RFC 7235 [2], using the *userID* and *password* assigned to the user. These authentication factors SHALL be passed directly in the HTTP header as an authorization grant to obtain a service access token to use for the subsequent API requests within the same session.

The OPTIONAL *rememberMe* parameter can be used, under the control of the user, in order to extend a successful authentication for subsequent sessions and to avoid the user to authenticate again within a predefined period of time. In this case, a refresh token will be obtained, which can be used in the *refresh_token* parameter in subsequent calls as an alternative to passing *userID* and *password* again for obtaining a new access token.

Note 23: The RECOMMENDED mechanism for service authorization is OAuth 2.0 (see [OAuth 2.0 Authorization](#)). HTTP Basic Authentication is an unsafe mechanism and therefore it SHOULD NOT be used, especially by signature application running as a service. It should only be used when there is a high degree of trust between the user and the signature application and when other authorization types like OAuth 2.0 are not available. This method may also be deprecated in future releases of this specification.

Input

The *userID* and *password* strings SHALL be encoded as defined in RFC 7235 [2] and provided in the HTTP Authorization header. If available, a refresh token MAY be alternatively used to re-authenticate the user after an access token has expired. This method allows the following parameters:

Parameter	Presence	Value	Description
<i>refresh_token</i>	REQUIRED Conditional	<i>String</i>	The long-lived refresh token returned from a previous call to this method with HTTP Basic Authentication. This MAY be used as an alternative to the Authorization header to reauthenticate the user according to the method described in RFC 6749 [11] par. 1.5. In such case the encoded <i>userID</i> and <i>password</i> SHALL not be provided in the HTTP Authorization header. NOTE: This refresh token MAY not be compatible with refresh tokens obtained by means of OAuth 2.0 authorization (see oauth2/token in oauth2/token).
<i>rememberMe</i>	OPTIONAL	<i>Boolean</i>	A boolean value typically corresponding to an option that the user may activate during the authentication phase to “stay signed in” and maintain a valid authentication across multiple sessions: <ul style="list-style-type: none"> • “true”: if the remote service supports user reauthentication, a <i>refresh_token</i> will be returned and the signature application may use it on a subsequent call to this method instead of passing an Authorization header. • “false”: a <i>refresh_token</i> will not be returned. If the parameter is omitted, it will default to “false”. This mechanism is based on the method described in RFC 6749 [11] section 1.5.
<i>clientData</i>	OPTIONAL	<i>String</i>	The <i>clientData</i> as defined in the Input parameter table in oauth2/authorize .

Output

This method returns the following values using the “application/json” format:

Attribute	Presence	Value	Description
<i>access_token</i>	REQUIRED	<i>String</i>	The short-lived service access token used to authenticate the subsequent API requests within the same session. This token SHALL be used as the value of the “Authorization: Bearer” in the HTTP header of the API requests. When receiving an API call with an expired token, the remote service SHALL return an error and require a new auth/login request.
<i>refresh_token</i>	OPTIONAL Conditional	<i>String</i>	The long-lived refresh token used to re-authenticate the user on the subsequent session. The value is returned if the <i>rememberMe</i> parameter in the request is “true” and the remote service supports user reauthentication. This mechanism is based on the method described in RFC 6749 [11] par. 1.5. NOTE: This <i>refresh_token</i> MAY not be compatible with refresh tokens obtained by means of OAuth 2.0 authorization.
<i>expires_in</i>	OPTIONAL	<i>Number</i>	The lifetime in seconds of the service access token. If omitted, the default expiration time is 3600 (1 hour).

Note 24: Access tokens and refresh tokens are credentials used to access protected resources. These tokens are strings representing a service authorization issued to the client. The strings MAY represent specific authorization criteria, but they SHOULD be opaque to the client.

Note 25: An existing refresh token MAY be automatically revoked if the user to whom it was issued performs a new service authorization with the *rememberMe* parameter set to “true”. It is up to the remote service to support a single or multiple refresh tokens per user.

Note 26: The lifetime of the *refresh_token* is determined by the RSSP.

Error Case	Status Code	Error	Error Description
------------	-------------	-------	-------------------

Error Case	Status Code	Error	Error Description
The authorization header does not match the basic HTTP authentication pattern ("Basic base64 ") - if refresh token is not present	401 (unauthorized)	invalid_request	Malformed authentication parameter.
Decoded credentials are not in the form "username:password"	400 (bad request)	invalid_request	Malformed username-password.
Invalid refresh_token parameter format	400 (bad request)	invalid_request	Invalid string parameter: refresh_token
Invalid refresh_token value	400 (bad request)	invalid_request	Invalid refresh_token
Authentication error – login failed	400 (bad request)	authentication_error	An error occurred during authentication process

Sample Request

```
POST /csc/v2/auth/login HTTP/1.1
Host: service.domain.org
Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ=
Content-Type: application/json

{
  "rememberMe": true
}
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/json"
-H "Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ="
-d '{"rememberMe": true}'
https://service.domain.org/csc/v2/auth/login
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
{
  "access_token": "4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA",
  "refresh_token":
    "_TiHRG-bA-H3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
  "expires_in": 3600
}
```

11.3 auth/revoke

Description

Revoke a service access token or refresh token that was obtained from the remote service or an associated authorization server. The revocation process is aligned with the OAuth 2.0 revocation

mechanism described in RFC 7009 [13] and can be applied to both tokens issued through calls to remote service methods (e.g. **auth/login** as defined in [auth/login](#)) and tokens issued as a result of an OAuth 2.0 flow (e.g. **oauth2/token** as defined in [oauth2/token](#)). This method MAY be used to enforce the security of the remote service. When the signature application needs to terminate a session, it is RECOMMENDED to invoke this method to prevent further access by reusing the token.

This method allows the signature application to invalidate its tokens according to the following approach:

- If the token passed to the request is a *refresh_token*, then the authorization server SHALL invalidate the refresh token and it SHALL also invalidate any existing access tokens based on the same authorization grant.
- If the token passed to the request is an *access_token*, then the authorization server SHALL invalidate the access token and it SHALL NOT revoke any existing refresh token based on the same authorization grant.

The invalidation of the token takes place immediately, and the token cannot be used again after its revocation. As a token issued in the process of credential authorization is automatically invalidated as soon as its usage limit is reached, a client does not have to revoke the corresponding token after use. However, a provider SHOULD support the revocation of such token type before reaching the usage limit.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
<i>token</i>	REQUIRED	<i>String</i>	The token that the signature application wants to get revoked.
<i>token_type_hint</i>	OPTIONAL	<i>String</i> <i>access_token</i> <i>refresh_token</i>	An OPTIONAL hint about the type of the token submitted for revocation. If the parameter is omitted, the remote service SHOULD try to identify the token across all the available tokens.
<i>clientData</i>	OPTIONAL	<i>String</i>	The <i>clientData</i> as defined in the Input parameter table in oauth2/authorize .

Output

This method has no output values and the response returns “No Content” status.

Error Case	Status Code	Error	Error Description
The authorization header does not match the pattern “Bearer [sessionKey]”	400 (bad request)	invalid_request	Malformed authorization header.
Missing or not String “token” parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter token
“token_hint” parameter present, not equal to “access_token” nor “refresh_token”	400 (bad request)	invalid_request	Invalid string parameter token_type_hint

Error Case	Status Code	Error	Error Description
Invalid access_token or refresh_token	400 (bad request)	invalid_request	Invalid string parameter token

Sample Request

```
POST /csc/v2/auth/revoke HTTP/1.1
Host: service.domain.org
Authorization: Bearer 4/CKN69L8gdSYp5_pwH3X1FQZ3ndFhkXf9P2_TiHRG-bA
Content-Type: application/json

{
  "token": "_TiHRG-bA-H3X1FQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
  "token_type_hint": "refresh_token",
  "clientData": "12345678"
}
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/json"
-H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3X1FQZ3ndFhkXf9P2_TiHRG-bA"
-d '{"token": "_TiHRG-bA-H3X1FQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
    "token_type_hint": "refresh_token",
    "clientData": "12345678"}'
https://service.domain.org/csc/v2/auth/revoke
```

Sample Response

```
HTTP/1.1 204 No Content
```

11.4 credentials/list

Description

Returns the list of credentials associated with a user identifier. A user MAY have one or multiple credentials hosted by a single remote signing service provider.

If requested, it can also return the signing certificate, the whole associated certificate chain, additional information about the signing certificate and/or information about the authorization mechanism required to authorize the access to the credentials for remote signing.

If the user is authenticated directly by the RSSP then the *userID* is implicit and SHALL NOT be specified.

This method can also be used in case of a community of users, to let the client retrieve the list of credentials assigned to a specific user of the community. In this case the *userID* SHALL be passed explicitly to retrieve the list of credentialIDs for a specific user. Managing a community of users that are authenticated by the client using a specific authentication framework is out of the scope of this specification.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
<i>userID</i>	REQUIRED Conditional	<i>String</i>	The identifier associated to the identity of the credential owner. This parameter SHALL NOT be present if the service authorization is user-specific (see NOTE below). In that case the <i>userID</i> is already implicit in the service access token passed in the Authorization header. If a user-specific service authorization is present, it SHALL NOT be allowed to use this parameter to obtain the list of credentials associated to a different user. The remote service SHALL return an error in such case.
<i>credentialInfo</i>	OPTIONAL	<i>Boolean</i>	Request to return the main information included in the public key certificate and the public key certificate itself or the certificate chain associated to the credentials. The default value is “false”, so if the parameter is omitted then the information will not be returned.
<i>certificates</i>	OPTIONAL Conditional	<i>String</i> none single chain	Specifies which certificates from the certificate chain SHALL be returned in <i>certs/certificates</i> . <ul style="list-style-type: none"> • “none”: No certificate SHALL be returned. • “single”: Only the end entity certificate SHALL be returned. • “chain”: The full certificate chain SHALL be returned. The default value is “single”, so if the parameter is omitted then the method will only return the end entity certificate(s). This parameter MAY be specified only if the parameter <i>credentialInfo</i> is “true”. If the parameter <i>credentialInfo</i> is not “true” and this parameter is specified its value SHALL be ignored.
<i>certInfo</i>	OPTIONAL Conditional	<i>Boolean</i>	Request to return various parameters containing information from the end entity certificate(s). This is useful in case the signature application wants to retrieve some details of the certificate(s) without having to decode it first. The default value is “false”, so if the parameter is omitted then the information will not be returned. This parameter MAY be specified only if the parameter <i>credentialInfo</i> is “true”. If the parameter <i>credentialInfo</i> is not “true” and this parameter is specified its value SHALL be ignored.
<i>authInfo</i>	OPTIONAL Conditional	<i>Boolean</i>	Request to return various parameters containing information on the authorization mechanisms supported by the corresponding credential (auth group). The default value is “false”, so if the parameter is omitted then the information will not be returned. This parameter MAY be specified only if the parameter <i>credentialInfo</i> is “true”. If the parameter <i>credentialInfo</i> is not “true” and this parameter is specified its value SHALL be ignored.
<i>onlyValid</i>	OPTIONAL Conditional	<i>Boolean</i>	Request to return only credentials usable to create a valid signature. The default value is “false”, so if the parameter is omitted then the method will return all credentials available to the owner. The remote service MAY NOT support this parameter. When the parameter is supported SHALL be returned in output.
<i>lang</i>	OPTIONAL	<i>String</i>	The <i>lang</i> as defined in the Input parameter table in info .
<i>clientData</i>	OPTIONAL	<i>String</i>	The <i>clientData</i> as defined in the Input parameter table in oauth2/authorize .

Note 27: User-specific service authorization include the following *authType*: “basic”, “digest” and “oauth2code”. Non-user-specific service authorization include the following *authType*: “external”, “TLS” or “oauth2client”.

Output

This method returns the following values using the “application/json” format:

Attribute	Presence	Value	Description
-----------	----------	-------	-------------

Attribute	Presence	Value	Description
<i>credentialIDs</i>	REQUIRED	<i>Array of String</i>	One or more credentialID(s) associated with the provided or implicit <i>userID</i> .
<i>credentialInfos</i>	OPTIONAL Conditional	<i>Array of CredentialInfo</i>	The contents of <i>credentialInfo</i> object are described below. If the <i>credentialInfo</i> parameter is not “true”, this value SHALL NOT be returned.
<i>onlyValid</i>	REQUIRED Conditional	<i>Boolean</i>	This value SHALL be returned true when the input parameter “onlyValid” was true, and the RSSP supports this feature, i.e. the RSSP only returns credentials which can be used for signing. If the values is false or the output parameter is omitted, then the list may contain credentials which cannot be used for signing.

The ‘credentialInfo Object’ is a JSON Object composed by the attributes specified in the following table.

Attribute	Presence	Value	Description
<i>credentialID</i>	REQUIRED	<i>String</i>	The credentialID identifying one of the credentials associated with the provided or implicit <i>userID</i> .
<i>description</i>	OPTIONAL	<i>String</i>	A free form description of the credential in the <i>lang</i> language. The maximum size of the string is 255 characters.
<i>signatureQualifier</i>	OPTIONAL	<i>String</i>	Identifier qualifying the type of signature this credential is suitable for (see signatures/signDoc).
<i>key/status</i>	REQUIRED	<i>String</i> enabled disabled	The status of the signing key of the credential: <ul style="list-style-type: none"> “enabled”: the signing key is enabled and can be used for signing. “disabled”: the signing key is disabled and cannot be used for signing. This MAY occur when the owner has disabled it or when the RSSP has detected that the associated certificate is expired or revoked.
<i>key/algo</i>	REQUIRED	<i>Array of String</i>	The list of OIDs of the supported key algorithms. For example: 1.2.840.113549.1.1.1 = RSA encryption, 1.2.840.10045.4.3.2 = ECDSA with SHA256.
<i>key/len</i>	REQUIRED	<i>Number</i>	The length of the cryptographic key in bits.
<i>key/curve</i>	REQUIRED Conditional	<i>String</i>	The OID of the ECDSA curve. The value SHALL only be returned if <i>keyAlgo</i> is based on ECDSA.
<i>cert/status</i>	OPTIONAL	<i>String</i> valid expired revoked suspended	The status of validity of the end entity certificate. The value is OPTIONAL, so the remote service SHOULD only return a value that is accurate and consistent with the actual validity status of the certificate at the time the response is generated.
<i>cert/certificates</i>	REQUIRED Conditional	<i>Array of String</i>	One or more Base64-encoded X.509v3 certificates from the certificate chain. If the <i>certificates</i> parameter is “chain”, the entire certificate chain SHALL be returned with the end entity certificate at the beginning of the array. If the <i>certificates</i> parameter is “single”, only the end entity certificate SHALL be returned. If the <i>certificates</i> parameter is “none”, this value SHALL NOT be returned.
<i>cert/issuerDN</i>	REQUIRED Conditional	<i>String</i>	The Issuer Distinguished Name from the X.509v3 end entity certificate as UTF-8-encoded character string according to RFC 4514 [4]. This value SHALL be returned when <i>certInfo</i> is “true”.
<i>cert/serialNumber</i>	REQUIRED Conditional	<i>String</i>	The Serial Number from the X.509v3 end entity certificate represented as hex-encoded string format. This value SHALL be returned when <i>certInfo</i> is “true”.
<i>cert/subjectDN</i>	REQUIRED Conditional	<i>String</i>	The Subject Distinguished Name from the X.509v3 end entity certificate as UTF-8-encoded character string, according to RFC 4514 [4]. This value SHALL be returned when <i>certInfo</i> is “true”.

Attribute	Presence	Value	Description
<i>cert/validFrom</i>	REQUIRED Conditional	<i>String</i>	The validity start date from the X.509v3 end entity certificate as character string, encoded as GeneralizedTime (RFC 5280 [8]) (e.g. "YYYYMMDDHHMMSSZ"). This value SHALL be returned when <i>certInfo</i> is "true".
<i>cert/validTo</i>	REQUIRED Conditional	<i>String</i>	The validity end date from the X.509v3 end entity certificate as character string, encoded as GeneralizedTime (RFC 5280 [8]) (e.g. "YYYYMMDDHHMMSSZ"). This value SHALL be returned when <i>certInfo</i> is "true".
<i>auth/mode</i>	REQUIRED	<i>String</i> explicit oauth2code	Specifies one of the authorization modes. For more information also see OAuth 2.0 Authorization : <ul style="list-style-type: none"> "explicit": the authorization process is managed by the signature application, which collects authentication factors of various types. "oauth2code": the authorization process is managed by the remote service using an OAuth 2.0 mechanism based on authorization code as described in Section 1.3.1 of RFC 6749 [11].
<i>SCAL</i>	OPTIONAL	<i>String</i> 1 2	Specifies if the RSSP will generate for this credential a signature activation data (SAD) or an access token with scope "credential" that contains a link to the hash to-be-signed: <ul style="list-style-type: none"> "1": The hash to-be-signed is not linked to the signature activation data. "2": The hash to-be-signed is linked to the signature activation data. <p>This value is OPTIONAL and the default value is "1". See Note below.</p>
<i>auth/expression</i>	OPTIONAL Conditional	<i>String</i>	An expression defining the combination of authentication objects required to authorize usage of the private key. If empty, an "AND" of all authentication objects is implied. Supported operators are: "AND" "OR" "XOR" "(" ")" This value SHALL NOT be returned if <i>auth/mode</i> is not "explicit".
<i>auth/objects</i>	REQUIRED Conditional	<i>Array of authentication object types</i>	The authentication object types available for this credential. This value SHALL only be returned if <i>auth/mode</i> is "explicit".
<i>multisign</i>	REQUIRED	<i>Number</i> ≥ 1	A number equal or higher to 1 representing the maximum number of signatures that can be created with this credential with a single authorization request (e.g. by calling <i>credentials/signHash</i> method, as defined in signatures/signHash , once with multiple hash values or calling it multiple times). The value of <i>numSignatures</i> specified in the authorization request SHALL NOT exceed the value of this value.
<i>lang</i>	OPTIONAL	<i>String</i>	The <i>lang</i> as defined in the Output parameter table in info .

Note 28: As described in the difference between SCAL1 and SCAL2 in [Credential authorization](#), the value "2" only gives information on the link between the hash and the SAD (or access token with scope "credential"), it does not give information if a full SCAL2 as described in CEN TS 119 241-1 [i.5] is implemented.

Error Case	Status Code	Error	Error Description
The authorization header does not match the pattern "Bearer [sessionKey]"	400 (bad request)	invalid_request	Malformed authorization header.

Error Case	Status Code	Error	Error Description
Not empty "userID" parameter in case of user-specific authorization	400 (bad request)	invalid_request	userID parameter MUST be null
Invalid "userID" format in case of no user-specific authorization	400 (bad request)	invalid_request	Invalid parameter userID
When present, invalid "certificates" parameter	400 (bad request)	invalid_request	Invalid parameter certificates

Sample Request

```
POST /csc/v2/credentials/list HTTP/1.1
Host: service.domain.org
Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA
Content-Type: application/json

{
  "credentialInfo": true,
  "certificates": "chain",
  "certInfo": true,
  "authInfo": true
}
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/json"
-H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
-d '{"credentialInfo": true,
    "certificates": "chain",
    "certInfo": true,
    "authInfo": true}'
https://service.domain.org/csc/v2/credentials/list
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "credentialIDs": [ "GX0112348", "HX0224685" ]
  "credentialInfos":
  [
    {
      "credentialID": "GX0112348",
      "key":
      {
        "status": "enabled",
        "algo": [ "1.2.840.113549.1.1.11", "1.2.840.113549.1.1.10" ],
        "len": 2048
      },
      "cert":
      {
        "status": "valid",
        "certificates":
        [
          "<Base64-encoded_X.509_end_entity_certificate>",

```

```

        "<Base64-encoded_X.509_intermediate_CA_certificate>",
        "<Base64-encoded_X.509_root_CA_certificate>"
    ],
    "issuerDN": "<X.500_issuer_DN_printable_string>",
    "serialNumber": "5AAC41CD8FA22B953640",
    "subjectDN": "<X.500_subject_DN_printable_string>",
    "validFrom": "20200101100000Z",
    "validTo": "20230101095959Z"
},
"auth": {
    "mode": "explicit",
    "expression": "PIN AND OTP",
    "objects": [
        {
            "type": "Password",
            "id": "PIN",
            "format": "N",
            "label": "PIN",
            "description": "Please enter the signature PIN"
        },
        {
            "type": "Password",
            "id": "OTP",
            "format": "N",
            "generator": "totp",
            "label": "Mobile OTP",
            "description": "Please enter the 6 digit code you received by
SMS"
        }
    ]
}
"multisign": 5,
"lang": "en-US"
},
{
    "credentialID": "HX0224685",
    .....
    .....
}
]
}

```

11.5 credentials/info

11.5.0.1 Description

Retrieves the credential. If requested, it can also return the signing certificate, the whole associated certificate chain, additional information about the signing certificate and/or information about the authorization mechanism required to authorize the access to the credential for remote signing.

11.5.0.2 Input

This method allows the following parameters:

Parameter	Presence	Value	Description
<i>credentialID</i>	REQUIRED	<i>String</i>	The unique identifier associated to the credential.
<i>certificates</i>	OPTIONAL	<i>String</i> none single chain	The <i>certificates</i> as defined in the Input parameter table in credentials/list .
<i>certInfo</i>	OPTIONAL	<i>Boolean</i>	The <i>certInfo</i> as defined in the Input parameter table in credentials/list .

Parameter	Presence	Value	Description
<i>authInfo</i>	OPTIONAL	<i>Boolean</i>	The <i>authInfo</i> as defined in the Input parameter table in credentials/list .
<i>lang</i>	OPTIONAL	<i>Strings</i>	The <i>lang</i> as defined in the Input parameter table in info .
<i>clientData</i>	OPTIONAL	<i>String</i>	The <i>clientData</i> as defined in the Input parameter table in oauth2/authorize .

11.5.0.3 Output:

This method returns the following values using the “application/json” format:

Attribute	Presence	Value	Description
<i>description</i>	OPTIONAL	<i>String</i>	The <i>description</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>signatureQualifier</i>	OPTIONAL	<i>String</i>	Identifier qualifying the type of signature this credential is suitable for (see signatures/signDoc).
<i>key/status</i>	REQUIRED	<i>String</i> enabled disabled	The <i>key/status</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>key/algo</i>	REQUIRED	<i>Array of String</i>	The <i>key/algo</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>key/len</i>	REQUIRED	<i>Number</i>	The <i>key/len</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>key/curve</i>	REQUIRED Conditional	<i>String</i>	The <i>key/curve</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/status</i>	OPTIONAL	<i>String</i> valid expired revoked suspended	The <i>cert/status</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/certificates</i>	REQUIRED Conditional	<i>Array of String</i>	The <i>cert/certificates</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/issuerDN</i>	REQUIRED Conditional	<i>String</i>	The <i>cert/issuerDN</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/serialNumber</i>	REQUIRED Conditional	<i>String</i>	The <i>cert/serialNumber</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/subjectDN</i>	REQUIRED Conditional	<i>String</i>	The <i>cert/subjectDN</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/validFrom</i>	REQUIRED Conditional	<i>String</i>	The <i>cert/validFrom</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>cert/validTo</i>	REQUIRED Conditional	<i>String</i>	The <i>cert/validTo</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>auth/mode</i>	REQUIRED	<i>String</i> explicit oauth2code	The <i>auth/mode</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>auth/expression</i>	OPTIONAL Conditional	<i>String</i>	The <i>auth/expression</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>auth/objects</i>	REQUIRED Conditional	<i>Array of authentication object types</i>	The <i>auth/objects</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>SCAL</i>	OPTIONAL	<i>String</i> 1 2	The <i>SCAL</i> as defined in the credentialInfo Object attribute table in credentials/list . See the Note in credentials/list about the <i>SCAL</i> attribute.
<i>multisign</i>	REQUIRED	<i>Number</i> ≥ 1	The <i>multisign</i> as defined in the credentialInfo Object attribute table in credentials/list .
<i>lang</i>	OPTIONAL	<i>String</i>	The <i>lang</i> as defined in the Output parameter table in info .

Error Case	Status Code	Error	Error Description
The authorization header does not match the pattern "Bearer [sessionKey]"	400 (bad request)	invalid_request	Malformed authorization header.
Missing or not String "credentialID" parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter credentialID
Invalid "credentialID" parameter	400 (bad request)	invalid_request	Invalid parameter credentialID
Invalid "certificates" parameter	400 (bad request)	invalid_request	Invalid parameter certificates

Sample Request

```
POST /csc/v2/credentials/info HTTP/1.1
Host: service.domain.org
Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA
Content-Type: application/json

{
  "credentialID": "GX0112348",
  "certificates": "chain",
  "certInfo": true,
  "authInfo": true
}
```

cURL example

```
curl -i -X POST
-H "Content-Type: application/json"
-H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
-d '{"credentialID": "GX0112348",
  "certificates": "chain",
  "certInfo": true,
  "authInfo": true }'
https://service.domain.org/csc/v2/credentials/info
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "key": {
    "status": "enabled",
    "algo": [
      "1.2.840.113549.1.1.1",
      "0.4.0.127.0.7.1.1.4.1.3"
    ],
    "len": 2048
  },
  "cert": {
    "status": "valid",
    "certificates": [
      "<Base64-encoded_X.509_end_entity_certificate>",
      "<Base64-encoded_X.509_intermediate_CA_certificate>",
      "<Base64-encoded_X.509_root_CA_certificate>"
    ]
  }
}
```