```
      ],
      "issuerDN":"<X.500_issuer_DN_printable_string>",
      "serialNumber":"5AAC41CD8FA22B953640",
      "subjectDN":"<X.500_subject_DN_printable_string>",
      "validFrom":"20180101100000Z",
      "validTo":"20190101095959Z"
   },
    "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"
}
```

11.6 credentials/authorize

Description

Authorize the access to the credential for remote signing, according to the authorization mechanisms associated to it. This method returns the Signature Activation Data (SAD) required to authorize the **signatures/signHash** method, as defined in signatures/signHash or **signatures/signDoc** method, as defined in signatures/signDoc.

Authentication objects and corresponding values collected from the user SHALL be included in the request according to the requirements specified by the **credentials/info** method, as defined in credentials/info.
This method SHALL be used in case of “explicit” authorization. This method SHALL also be used in case that no authentication objects are required, to trigger a possible authorization mechanism managed by the remote service. This method SHALL NOT be used in case of “oauth2” credential authorization; instead, any of the available OAuth 2.0 authorization mechanisms SHALL be used.

The *numSignatures* parameter SHALL indicate the total number of signatures to authorize. In case of multi-signature transactions where the **signatures/signHash** method is invoked multiple times, the signature application SHALL obtain a new SAD by invoking the **credentials/extendTransaction** method, as defined in credentials/extendTransaction, before the current SAD expires. In such cases the hashes to be signed may not all be available when the authorization is performed, for example in case of multiple signatures applied to a PDF file with a single credential. Further hashes should then be passed as an input to **credentials/extendTransaction** to make each SAD calculation dependent on the data to be signed. This approach may break the support of SCAL 2 requirements, therefore a remote signing service MAY fail if the *hash* parameter does not contain a number of hash values corresponding to the value in *numSignatures*.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
credentialID	REQUIRED	*String*	The *credentialID* as defined in the Input parameter table in credentials/info.
numSignatures	REQUIRED	*Number*	The number of signatures to authorize. Multi-signature transactions can be obtained by using a combination of passing an array of hash values and calling the **signatures/signHash** method, as defined in signatures/signHash, multiple times.
hashes	REQUIRED Conditional	*Array of String*	One or more Base64-encoded hash values to be signed. It allows the server to bind the *SAD* to the hash(es), thus preventing an authorization to be used to sign a different content. If the *SCAL* parameter returned by **credentials/info** method, as defined in credentials/info, for the current credentialID is "2" the *hash* parameter SHALL be used and the number of hash values SHOULD correspond to the value in *numSignatures*. If the SCAL parameter is "1", the *hash* parameter is OPTIONAL.
hashAlgorithmOID	REQUIRED Conditional	*String*	String containing the OID of the hash algorithm used to generate the hashes.
authData	REQUIRED Conditional	*Array of authentication objects*	The authentication objects as described by the authentication object types in **credentials/info**. It SHALL be used only when *auth/mode* from **credentials/info** is "explicit".
description	OPTIONAL	*String*	A free form description of the authorization transaction in the *lang* language. The maximum size of the string is 500 characters. It can be useful to provide some hints about the occurring transaction.
clientData	OPTIONAL	*String*	The *clientData* as defined in the Input parameter table in oauth2/authorize.

Output

With HTTP status code 200, the method returns the Signature Activation Data using the "application/json" format:

Attribute	Presence	Value	Description
SAD	REQUIRED	*String*	The Signature Activation Data (SAD) to be used as input to the **signatures/signHash** method, as defined in signatures/signHash.
expiresIn	OPTIONAL	*Number*	The lifetime in seconds of the SAD. If omitted, the default expiration time is 3600 (1 hour).

With HTTP status code 202 the method indicates that some authorization is still underway. The result contains a handle that can be used to poll the state of the authorization via **credentials/authorizeCheck**.

Attribute	Presence	Value	Description
handle	REQUIRED	*String*	An opaque handle that can be used to request the state of the authorization.

Error Case	Status Code	Error	Error Description

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
Signing key for “credentialID” is disabled	400 (bad request)	invalid_request	The credential identified by credentialID is disabled
Missing or not integer “numSignatures” parameter	400 (bad request)	invalid_request	Missing (or invalid type) integer parameter numSignatures
“numSignatures” < 1	400 (bad request)	invalid_request	Invalid value for parameter numSignatures
“numSignatures” > “multisign”	400 (bad equrest)	invalid_request	Numbers of signatures is too high
Invalid authentication data	400 (bad request)	invalid_authentication_data	The authentication data is invalid
Credential locked	400 (bad request)	invalid_request	Credential locked

Note 29: In case wrong authentication data is provided several times, the remote signing service MAY lock the credential or the usage of respective authentication objects. The policy adopted by the RSSP in this regard is out of the scope of this specification.

Sample Request

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

{
   "credentialID":"GX0112348",
   "numSignatures":2,
   "hashes":[
      "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
      "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
   ],
   "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
   "authData": [
        {
            "id": "PIN",
            "value": "123456"
        },
        {
            "id": "OTP",
            "value": "738496"
        }
   ],
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "credentialID": "GX0112348",
           "numSignatures": 2,
           "hashes": [ "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
           "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
           ],
       "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
           "authData": [
                {
                    "id": "PIN",
                    "value": "123456"
                },
                {
                    "id": "OTP",
                    "value": "738496"
                }
           ],
           "clientData": "12345678" }'
     https://service.domain.org/csc/v2/credentials/authorize
```

Sample Response

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

{
   "SAD":"_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw"
}
```

Sample Response

```
HTTP/1.1 202 OK
Content-Type: application/json;charset=UTF-8

{
   "handle": "878287f37b2bv293bv2bv237bv297bvbv"
}
```

11.7 credentials/authorizeCheck

Description

After a credentials/authorize with HTTP result code 202, the client may use the handle returned to poll the authorization state.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
handle	REQUIRED	*String*	The handle value returned from **credentials/authorize**.

Output

With HTTP status code 200, the method returns the Signature Activation Data using the “application/json” format:

Attribute	Presence	Value	Description
SAD	REQUIRED	*String*	The Signature Activation Data (SAD) to be used as input to the **signatures/signHash** method, as defined in signatures/signHash.
expiresIn	OPTIONAL	*Number*	The lifetime in seconds of the SAD. If omitted, the default expiration time is 3600 (1 hour).

With HTTP status code 202 the method indicates that some authorization is still underway. The result contains a handle that can be used to poll the state of the authorization via repeated calls to **credentials/authorizeCheck**.

Attribute	Presence	Value	Description
handle	REQUIRED	*String*	An opaque handle that can be used to request the state of the authorization.

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.
Invalid “handle” parameter	400 (bad request)	invalid_request	Invalid parameter handle
Invalid authentication data	400 (bad request)	invalid_authentication_data	The authentication data is invalid
Credential locked	400 (bad request)	invalid_request	Credential locked

Note 30: In case wrong authentication data is provided several times, the remote signing service MAY lock the credential or the usage of respective authentication objects. The policy adopted by the RSSP in this regard is out of the scope of this specification.

Sample Request

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

{
   "handle":"878287f37b2bv293bv2bv237bv297bvbv"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "handle":"878287f37b2bv293bv2bv237bv297bvbv" }'
     https://service.domain.org/csc/v2/credentials/authorizeCheck
```

Sample Response

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

{
   "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw"
}
```

Sample Response

```
HTTP/1.1 202 OK
Content-Type: application/json;charset=UTF-8

{
   "handle": "878287f37b2bv293bv2bv237bv297bvbv"
}
```

11.8 credentials/getChallenge

Description

Get a challenge for the referenced authentication object.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
credentialID	REQUIRED	*String*	The identifier associated to the credential.
authObjectID	REQUIRED	*String*	The identifier of the authentication object we need a challenge for.

Output

With HTTP status code 200, the method returns a challenge:

Attribute	Presence	Value	Description
challenge	REQUIRED	*String*	The authentication object challenge.

With HTTP status code 204, the method indicates that a challenge has been sent by out of band means, returning no output values.

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
Invalid “credentialID” parameter	400 (bad request)	invalid_request	Invalid parameter credentialID
Invalid “authObjectID” parameter	400 (bad request)	invalid_request	Invalid parameter authObjectID

Sample I - in-band challenge

Sample Request

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

{
   "credentialID": "GX0112348",
   "authObjectID": "fallback question"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{
        "credentialID": "GX0112348",
        "authObjectID": "fallback question"
     }'
     https://service.domain.org/csc/v2/credentials/getChallenge
```

Sample Response

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

{
   "challenge": "What's your mother's birth name?"
}
```

Sample II - out-of-band challenge

Sample Request

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

{
   "credentialID": "GX0112348",
   "authObjectID": "OTP"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{
        "credentialID": "GX0112348",
        "authObjectID": "OTP"
     }'
     https://service.domain.org/csc/v2/credentials/getChallenge
```

Sample Response

```
HTTP/1.1 204 OK
```

11.9 credentials/extendTransaction

Description

Extends the validity of a multi-signature transaction authorization by obtaining a new Signature Activation Data (SAD). This method SHALL be used in case of multi-signature transaction when the API method **signatures/signHash**, as defined in signatures/signHash, is invoked multiple times with a single credential authorization event.
It can also be used to renew a SAD, before it expires, when signature operations take longer than allowed by the *expiresIn* value. Expired SAD cannot be extended.

The RSSP SHALL invalidate the SAD when the number of authorized signatures, specified with *numSignatures* in the credential authorization event, has been created.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
credentialID	REQUIRED	*String*	The *credentialID* as defined in the Input parameter table in credentials/info.
hashes	REQUIRED Conditional	*Array of String*	One or more Base64-encoded hash values to be signed. It allows the server to bind the new *SAD* to the hash, thus preventing an authorization to be used to sign a different content. It SHALL be used if the *SCAL* parameter returned by **credentials/info**, as defined in credentials/info, for the current *credentialID* is “2” , otherwise it is OPTIONAL.
hashAlgorithmOID	REQUIRED Conditional	*String*	String containing the OID of the hash algorithm used to generate the hashes.
SAD	REQUIRED	*String*	The current unexpired Signature Activation Data. This token is returned by the **credentials/authorize**, as defined in credentials/authorize, or by the previous call to **credentials/extendTransaction**.
clientData	OPTIONAL	*String*	The *clientData* as defined in the Input parameter table in oauth2/authorize.

Note 31: This method can be used for applying multiple signatures to a PDF document from a single user, e.g. to sign separately different parts of the document, with a single credential authorization event. The PDF standard adopts nested signatures so the hashes for multiple signatures can only be calculated after the previous signature has been created. This method allows to calculate a new SAD based on new hash values that were not available when the credential authorization event occurred. The sequence diagram in Create a remote multi-signatures transaction with a PDF document shows this use case.

Output

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

Attribute	Presence	Value	Description
SAD	REQUIRED	*String*	The new Signature Activation Data required to sign multiple times with a single authorization.
expiresIn	OPTIONAL	*Number*	The lifetime in seconds of the SAD. If omitted, the default expiration time is 3600 (1 hour).

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.

Note 32: In case a wrong PIN or OTP is provided several times, the remote signing service MAY lock the credential or the usage of the PIN or OTP. The policy adopted by the RSSP in this regard is out of the scope of this specification.

Sample Request

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

{
   "credentialID":"GX0112348",
   "hashes":[
      "WlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
   ],
   "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
   "SAD":"_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "credentialID": "GX0112348",
           "hashes": [ "WlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0=" ],
       "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
           "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
           "clientData": "12345678" }'
     https://service.domain.org/csc/v2/credentials/extendTransaction
```

Sample Response

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

{
    "SAD":"1/UsHDJ98349h9fgh9348hKKHDkHWVkl/8hsAW5usc8_5="
}
```

11.10 signatures/signHash

Description

Calculate the remote digital signature of one or multiple hash values provided in input.

This method requires service and credential authorization.

The signing application MUST pass an access token with scope “service” or “credential” in the “Authorization” HTTP header as defined in RFC 6750 [12].

If the credential authorization mode is “explicit”, the signing application MUST pass Signature Activation Data (SAD) in the `SAD` request parameter (see below). SAD may be obtained from **credential/authorize**.

If the credential authorization mode is “oauth2code” and the access token passed in the “Authorization” HTTP header has scope “service”, the signing application MUST pass an access token with scope “credential” in the `SAD` request parameter. This is not required, if the the access token passed in the “Authorization” HTTP header has scope “credential”.

In case of multi-signature transactions, the SAD SHALL be updated with **credentials/extendTransaction**, as defined in credentials/extendTransaction, every time this method is invoked until the maximum number of authorized signatures has been generated.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
credentialID	REQUIRED	*String*	The credentialID as defined in the Input parameter table in credentials/info.
SAD	REQUIRED Conditional	*String*	The Signature Activation Data returned by the Credential Authorization methods. Not needed if the signing application has passed an access token in the “Authorization” HTTP header with scope “credential”, which is also good for the credential identified by `credentialID`. Note: For backward compatibility, signing applications MAY pass access tokens with scope “credential” in this parameter.
hashes	REQUIRED	*Array of String*	One or more hash values to be signed. This parameter SHALL contain the Base64-encoded raw message digest(s).
hashAlgorithmOID	REQUIRED Conditional	*String*	The OID of the algorithm used to calculate the hash value(s). This parameter SHALL be omitted or ignored if the hash algorithm is implicitly specified by the *signAlgo* algorithm. Only hashing algorithms as strong or stronger than SHA256 SHALL be used. The hash algorithm SHOULD follow the recommendations of ETSI TS 119 312 [21].
signAlgo	REQUIRED	*String*	The OID of the algorithm to use for signing. It SHALL be one of the values allowed by the credential as returned in *keyAlgo* by the **credentials/info** method, as defined in credentials/info or by **credentials/list** method, as defined in credentials/list.
signAlgoParams	REQUIRED Conditional	*String*	The Base64-encoded DER-encoded ASN.1 signature parameters, if required by the signature algorithm. Some algorithms like RSASSA-PSS, as defined in RFC 8017 [18], may require additional parameters.

Parameter	Presence	Value	Description
operationMode	OPTIONAL	*String*	The type of operation mode requested to the remote signing server. It SHALL take one of the following values: • "A": an asynchronous operation mode is requested. • "S": a synchronous operation mode is requested. The default value is "S", so if the parameter is omitted then the remote signing server will manage the request in synchronous operation mode.
validity_period	OPTIONAL Conditional	*Integer*	Maximum period of time, expressed in milliseconds, until which the server SHALL keep the request outcome(s) available for the client application retrieval. This parameter MAY be specified only if the parameter *operationMode* is "A". If the parameter *operationMode* is not "A" and this parameter is specified its value SHALL be ignored. The RSSP SHOULD define in its service policy the default and maximum values of this parameter. If the RSSP does not define in its service policy any default and maximum values of this parameter it means that any value MAY be passed in this parameter.
response_uri	OPTIONAL Conditional	*String*	Value of one location where the server will notify the signature creation operation completion, as an URI value. This parameter MAY be specified only if the parameter *operationMode* is "A". If the parameter *operationMode* is not "A" and this parameter is specified its value SHALL be ignored. If the parameter operationMode is "A" and this parameter is omitted then the remote signing server will not make any notification.
clientData	OPTIONAL	*String*	The *clientData* 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
signatures	REQUIRED Conditional	*Array of String*	One or more Base64-encoded signed hash(s). In case of multiple signatures, the signed hashes SHALL be returned in the same order as the corresponding hashes provided as an input parameter. This value SHALL be returned when *operationMode* is not "A".
responseID	REQUIRED Conditional	*String*	Arbitrary string value generated by the server uniquely identifying the response originated from the server itself. This value SHALL be returned when *operationMode* is "A".

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 "SAD" parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter SAD
Invalid "SAD" parameter	400 (bad request)	invalid_request	Invalid parameter SAD
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

Error Case	Status Code	Error	Error Description
Missing or not Array “hash” parameter	400 (bad request)	invalid_request	Missing (or invalid type) array parameter hash
Empty hash parameter	400 (bad request)	invalid_request	Empty hash array
Invalid Base64 hash element	400 (bad request)	invalid_request	Invalid Base64 hash string parameter
Unauthorized hash	400 (bad request)	invalid_request	Hash is not authorized by the SAD.
Missing or not String “signAlgo” parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter signAlgo
Missing or not String “signAlgoParams” parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter signAlgoParams
Missing or not String “hashAlgorithmOID” parameter when “signAlgo” is equal to “1.2.840.113549 .1.1.1”	400 (bad request)	invalid_request	Missing (or invalid type) string parameter hashAlgorithmOID
Invalid “hashAlgorithmOID” parameter	400 (bad request)	invalid_request	Invalid parameter hashAlgorithmOID
Invalid “signAlgo” parameter	400 (bad request)	invalid_request	Invalid parameter signAlgo
When present, invalid “operationMode” parameter	400 (bad request)	invalid_request	Invalid parameter operationMode
When present, invalid “validity_period” parameter	400 (bad request)	invalid_request	Invalid parameter validity_period
When present, out of bounds “validity_period” parameter	400 (bad request)	invalid_request	Out of bounds parameter validity_period
When present, invalid “response_uri” parameter	400 (bad request)	invalid_request	Invalid parameter response_uri
When present, invalid “clientData” format (not string)	400 (bad request)	invalid_request	Invalid parameter clientData
Invalid “hashes” length	400 (bad request)	invalid_request	Invalid digest value length
The OTP used to generate the “SAD” is invalid	400 (bad request)	invalid_otp	The OTP is invalid
Expired “SAD”	400 (bad request)	invalid_request	SAD expired

Error Case	Status Code	Error	Error Description
Expired credential	400 (bad request)	invalid_request	Signing certificate 'O=[organization],CN=[comm on_name]' is expired.

Sample Request

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

{
   "credentialID":"GX0112348",
   "SAD":"_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
   "hashes":[
      "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
      "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
   ],
   "hashAlgorithmOID":"2.16.840.1.101.3.4.2.1",
   "signAlgo":"1.2.840.113549.1.1.1",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer
     4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "credentialID": "GX0112348",
           "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
           "hashes": [ "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
           "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
           ],
           "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
           "signAlgo": "1.2.840.113549.1.1.1",
           "clientData": "12345678"}'
     https://service.domain.org/csc/v2/signatures/signHash
```

Sample Response

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

{
   "signatures":
   [
      "KedJuTob5gtvYx9qM3k3gm7kbLBwVbEQRl26S2tmXjqNND7MRGtoew==",
      "Idhef7xzgtvYx9qM3k3gm7kbLBwVbE98239S2tm8hUh85KKsfdowel=="
   ]
}
```

Sample Request

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

```
{
   "credentialID":"GX0112348",
   "SAD":"_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
   "hashes":[
      "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
      "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
   ],
   "hashAlgorithmOID":"2.16.840.1.101.3.4.2.1",
   "signAlgo":"1.2.840.113549.1.1.1",
   "operationMode": "A",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer
     4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "credentialID": "GX0112348",
           "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
           "hashes": [ "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
           "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
           ],
           "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
           "signAlgo": "1.2.840.113549.1.1.1",
           "operationMode": "A",
           "clientData": "12345678"}'
     https://service.domain.org/csc/v2/signatures/signHash
```

Sample Response

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

{
   "responseID":"158112-652341-khj"
}
```

Sample Request

```
POST /csc/v2/signatures/signHash HTTP/1.1
Host: service.domain.org
Content-Type: application/json
Authorization: Bearer 5/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA

{
   "credentialID":"GX0112348",
   "hashes":[
      "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
      "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNEllaXFsRGcvSFd1ST0="
   ],
   "hashAlgorithmOID":"2.16.840.1.101.3.4.2.1",
   "signAlgo":"1.2.840.113549.1.1.1",
   "operationMode": "A",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer
     5/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "credentialID": "GX0112348",
           "hashes": [ "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
           "c1RPZ3dPbSs0NzRnRmowcTB4MWlTTnNwS3FiY3NlNElla XFsRGcvSFd1ST0="
           ],
           "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
           "signAlgo": "1.2.840.113549.1.1.1",
           "operationMode": "A",
           "clientData": "12345678"}'
     https://service.domain.org/csc/v2/signatures/signHash
```

Sample Response

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

{
   "responseID":"158112-652341-khj"
}
```

11.11 signatures/signDoc

Description

Create one or more AdES signatures. Either the documents to be signed or the SDRs (in this specification it is intended to be the hash values of the documents to be signed) SHALL be provided to the method. An AdES signature will be created for each of these input components. Other components are used to select the type of signature that will be created for each document or document representation.

This method requires service and credential authorization as defined in signatures/signHash.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
credentialID	REQUIRED Conditional	*String*	The credentialID as defined in the Input parameter table in credentials/info. At least one of the two values *credentialID* and *signatureQualifier* SHALL be present. Both values MAY be present.
signatureQualifier	REQUIRED Conditional	*String*	Identifier of the signature type to be created, e.g. "eu_eidas_qes" to denote a Qualified Electronic Signature according to eIDAS. This specification defines a set of such identifiers (see table below), service providers can also define and use their own identifiers. At least one of the two values *credentialID* and *signatureQualifier* SHALL be present. Both values MAY be present.
SAD	REQUIRED Conditional	*String*	The Signature Activation Data returned by the Credential Authorization methods. Not needed if the signing application has passed an access token with scope "credential" in the "Authorization" HTTP header, which is also good for the credential identified by `credentialID` or the signature qualifier identified by `signatureQualifier`. Note: For backward compatibility, signing applications MAY pass access tokens with scope "credential" in this parameter.

Parameter	Presence	Value	Description
documentDigests	REQUIRED Conditional	*JSON Array*	An array containing JSON objects containing a hash value representing one or more SDRs, the respective digest algorithm OID used to calculate this hash value and further request parameters (see below). This parameter or the parameter *documents* MUST be present in a request. Otherwise the method SHALL return an error condition.
documents	REQUIRED Conditional	*JSON array*	An array containing JSON objects, each of them containing a base64-encoded document content to be signed and further request parameter. This parameter or the parameter *documentDigests* MUST be present in a request. Otherwise the method SHALL return an error condition.
operationMode	OPTIONAL	*String*	The *operationMode* as defined in the Input parameter table in signatures/signHash.
validity_period	OPTIONAL Conditional	*Integer*	The *validity_period* as defined in the Input parameter table in signatures/signHash.
response_uri	OPTIONAL Conditional	*String*	The *response_uri* as defined in the Input parameter table in signatures/signHash.
clientData	OPTIONAL	*String*	The *clientData* as defined in the Input parameter table in oauth2/authorize.
returnValidationInfo	OPTIONAL	*Boolean*	This parameter SHALL be set to “true” to request the service to return the “validationInfo” as defined below. The default value is “false”, i.e. no “validationInfo” info is provided. This parameter SHALL be supported in conjunction with “signature_format” “P”, “conformance_level” “AdES-B-LT” and use of “documentDigests”. For all other cases, the *info* methods states if this feature is supported or not.

This table lists the pre-defined signature qualifier to be used in conjunction with the `signatureQualifier` parameter.

Note 33: *signatureQualifiers follow the syntax X_Y_Z (e.g. eu_eidas_qes) where: X: The ISO 3166-1 [22] Alpha-2 code of the Country where the signature legislation is defined (e.g. eu for Europe). Y: The shortform name of the legislation (e.g. eidas for Electronic Identification And Trust Services) Z: The shortform name of the signature type defined by the legislation (e.g. qes for Qualified Electronic Signatures)

Identifier	Description
eu_eidas_qes	This identifier refers to a qualified electronic signature under eIDAS.
eu_eidas_aes	This identifier refers to an advanced electronic signature under eIDAS.
eu_eidas_aesqc	This identifier refers to an advanced electronic signature with qualified certificate under eIDAS.
eu_eidas_qeseal	This identifier refers to a qualified electronic seal under eIDAS.
eu_eidas_aeseal	This identifier refers to an advanced electronic seal under eIDAS.
eu_eidas_aesealqc	This identifier refers to an advanced electronic seal with qualified certificate under eIDAS.
za_ecta_aes	This identifier refers to an advanced electronic signature defined by the South African ECT Act
za_ecta_oes	This identifier refers to an ordinary electronic signature defined by the South African ECT Act

The `documentDigests` parameter is a JSON array composed of JSON Object composed by the following parameters:

- `hashes`
- `hashAlgorithmOID`
- `signature_format`
- `conformance_level`
- `signAlgo`

- `signAlgoParams`
- `signed_envelope_property`

specified according to the following table.

Parameter	Presence	Value	Description
hashes	REQUIRED Conditional	*Array of String*	One or more hash values representing one or more SDRs. This parameter SHALL contain the Base64-encoded hash(es) of the documents to be signed. In case a hashes were provided for the credential authorization, then the RSSP SHALL verify that each of the hashes in this parameter corresponds to one of the hashes provided in the credential authorization.
hashAlgorithmOID	REQUIRED Conditional	*String*	Hashing algorithm OID used to calculate document(s) hash(es). This parameter MAY be omitted or ignored if the hash algorithm is implicitly specified by the *signAlgo* algorithm. Only hashing algorithms as strong or stronger than SHA256 SHALL be used. The hash algorithm SHOULD follow the recommendations of ETSI TS 119 312 [21].
signature_format	REQUIRED	*String*	The required signature format: • "C" SHALL be used to request the creation of a CAdES signature; • "X" SHALL be used to request the creation of a XAdES signature. • "P" SHALL be used to request the creation of a PAdES signature. • "J" SHALL be used to request the creation of a JAdES signature.
conformance_level	OPTIONAL	*String*	The required signature conformance level: • "Ades-B-B" SHALL be used to request the creation of a a baseline 191x2 level B signature; • "Ades-B-T" SHALL be used to request the creation of a a baseline 191x2 level T signature; • "Ades-B-LT" SHALL be used to request the creation of a a baseline 191x2 level LT signature; • "Ades-B-LTA" SHALL be used to request the creation of a a baseline 191x2 level LTA signature; • "Ades-B" SHALL be used to request the creation of a a baseline etsits level B signature; • "Ades-T" SHALL be used to request the creation of a a baseline etsits level T signature; • "Ades-LT" SHALL be used to request the creation of a a baseline etsits level LT signature; • "Ades-LTA" SHALL be used to request the creation of a a baseline etsits level LTA signature. The parameter is optional. The default level is AdES-B-B in case it is omitted. If a timestamp is needed its request and inclusion is managed by the signing server according to signing server configuration and policies.
signAlgo	REQUIRED	*String*	The *signAlgo* as defined in the Input parameter table in signatures/signHash. If the parameter *hashAlgorithmOID* defined in the *documentDigests* Object is passed and is in contradiction with the value of this parameter *signAlgo* the method SHALL return an error condition.
signAlgoParams	REQUIRED Conditional	*String*	The *signAlgoParams* as defined in the Input parameter table in signatures/signHash.

Parameter	Presence	Value	Description
signed_props	OPTIONAL	*Array of attribute*	List of signed attributes. The attributes that may be included depend on the signature format and the signature creation policy. The contents of *attribute* object are described below.

signed_envelope_property	OPTIONAL Conditional	*String*	The required property concerning the signed envelope whose possible values depend on the value of the *signature_format* parameter.
			According to the type of selected *signature_format* a client application may specify the following signature properties.
			
			• CAdES
			  ◦ Detached
			  ◦ Attached
			  ◦ Parallel
			• PAdES
			  ◦ Certification
			  ◦ Revision
			• XAdES
			  ◦ Enveloped
			  ◦ Enveloping
			  ◦ Detached
			• JAdES
			  ◦ Detached
			  ◦ Attached
			  ◦ Parallel
			
			The default values are the following ones.
			
			• CAdES
			  ◦ Attached
			• PAdES
			  ◦ Certification
			• XAdES
			  ◦ Enveloped
			• JAdES
			  ◦ Attached

The `documents` parameter is a JSON array composed of JSON Object composed by the following parameters:

- `document`
- `signature_format`
- `conformance_level`
- `signAlgo`
- `signAlgoParams`
- `signed_envelope_property`

specified according to the following table.

Parameter	Presence	Value	Description
document	REQUIRED<	*String*	base64-encoded document content to be signed. In case a hashes were provided for the credential authorization, then the RSSP SHALL verify that the hash of the document in this parameter corresponds to one of the hashes provided in the credential authorization.

Parameter	Presence	Value	Description
signature_format	REQUIRED	*String*	The required signature format: • “C” SHALL be used to request the creation of a CAdES signature; • “X” SHALL be used to request the creation of a XAdES signature. • “P” SHALL be used to request the creation of a PAdES signature. • “J” SHALL be used to request the creation of a JAdES signature.
conformance_level	OPTIONAL	*String*	The required signature conformance level: • “Ades-B-B” SHALL be used to request the creation of a a baseline 191x2 level B signature; • “Ades-B-T” SHALL be used to request the creation of a a baseline 191x2 level T signature; • “Ades-B-LT” SHALL be used to request the creation of a a baseline 191x2 level LT signature; • “Ades-B-LTA” SHALL be used to request the creation of a a baseline 191x2 level LTA signature; • “Ades-B” SHALL be used to request the creation of a a baseline etsits level B signature; • “Ades-T” SHALL be used to request the creation of a a baseline etsits level T signature; • “Ades-LT” SHALL be used to request the creation of a a baseline etsits level LT signature; • “Ades-LTA” SHALL be used to request the creation of a a baseline etsits level LTA signature. The parameter is optional. The default level is AdES-B-B in case it is omitted. If a timestamp is needed its request and inclusion is managed by the signing server according to signing server configuration and policies.
signAlgo	REQUIRED	*String*	The *signAlgo* as defined in the Input parameter table in signatures/signHash. If the parameter *hashAlgorithmOID* defined in the *documentDigests* Object is passed and is in contradiction with the value of this parameter *signAlgo* the method SHALL return an error condition.
signAlgoParams	REQUIRED Conditional	*String*	The *signAlgoParams* as defined in the Input parameter table in signatures/signHash.
signed_props	OPTIONAL	*Array of attribute*	List of signed attributes. The attributes that may be included depend on the signature format and the signature creation policy. The contents of *attribute* object are described below.

Parameter	Presence	Value	Description
signed_envelope_property	OPTIONAL Conditional	*String*	The required property concerning the signed envelope whose possible values depend on the value of the *signature_format* parameter. According to the type of selected *signature_format* a client application may specify the following signature properties. • CAdES ◦ Detached ◦ Attached ◦ Parallel • PAdES ◦ Certification ◦ Revision • XAdES ◦ Enveloped ◦ Enveloping ◦ Detached • JAdES ◦ Detached ◦ Attached ◦ Parallel The defaul values are the following ones. • CAdES ◦ Attached • PAdES ◦ Certification • XAdES ◦ Enveloped • JAdES ◦ Attached

The 'attribute' is a JSON Object composed by the following attributes:

- `attribute_name`
- `attribute_value`

specified according to the following table.

Parameter	Presence	Value	Description
attribute_name	REQUIRED	*String*	Name or OID of the attribute/property to be included in the signature. Below the table a list of the attributes/properties names that can be referenced in this component in order to request the inclusion of the corresponding signed attributes/properties in the signature. Other attributes and/or properties whose names are defined in the table in clause 6.3 of ETSI EN 319 122-1 [29], ETSI EN 319 132-1 [30], ETSI EN 319 142-1 [31], ETSI TS 119 182-1 [32] documents may be supported by the signing server.
attribute_value	REQUIRED Conditional	*String*	Depending on the attribute/property specified in the *attribute_name* parameter, this parameter contains the value to be used for such attribute/property to be included in the signature. When some element of this parameter is not defined the signing server SHALL calculate it, if needed.

As an alternative to the attributes/properties names listed in the table below it is also possible using the corresponding attributes/properties oids.

attribute_name	attribute_value