attribute_name	attribute_value
commitment-type-indication	This parameter contains the Base64-encoding of the attribute commitment-type-indication defined in clause 5.2.3 of ETSI EN 319 122-1 [29].
content-hints	This parameter contains the Base64-encoding of the attribute content-hints defined in clause 5.2.4.1 of ETSI EN 319 122-1 [29].
mime-type	This parameter contains the Base64-encoding of the attribute mime-type defined in clause 5.4.2.2 of ETSI EN 319 122-1 [29].
signer-location	This parameter contains the Base64-encoding of the attribute signer-location defined in clause 5.2.5 of ETSI EN 319 122-1 [29].
content-time-stamp	This parameter contains the Base64-encoding of the attribute content-time-stamp defined in clause 5.2.8 of ETSI EN 319 122-1 [29].
signer-attributes-v2	This parameter contains the Base64-encoding of the attribute signer-attributes-v2 defined in clause 5.2.6.1 of ETSI EN 319 122-1 [29].
signature-policy-identifier	This parameter contains the Base64-encoding of the attribute signature-policy-identifier defined in clause 5.2.9.1 of ETSI EN 319 122-1 [29].
content-reference	This parameter contains the Base64-encoding of the attribute content-reference defined in clause 5.2.11 of ETSI EN 319 122-1 [29].
content-identifier	This parameter contains the Base64-encoding of the attribute content-identifier defined in clause 5.2.12 of ETSI EN 319 122-1 [29].
Location	This parameter contains the Base64-encoding of the attribute Location defined in clause 5.3 of ETSI EN 319 142-1 [31].
Reason	This parameter contains the Base64-encoding of the attribute Reason defined in clause 5.3 of ETSI EN 319 142-1 [31].
Name	This parameter contains the Base64-encoding of the attribute Name defined in clause 5.3 of ETSI EN 319 142-1 [31].
ContactInfo	This parameter contains the Base64-encoding of the attribute ContactInfo defined in clause 5.3 of ETSI EN 319 142-1 [31].
SignerRoleV2	This parameter contains the Base64-encoding of the attribute SignerRoleV2 defined in clause 5.2.6 of ETSI EN 319 132-1 [30].
CommitmentTypeIndication	This parameter contains the Base64-encoding of the attribute CommitmentTypeIndication defined in clause 5.2.3 of ETSI EN 319 132-1 [30].
SignatureProductionPlaceV2	This parameter contains the Base64-encoding of the attribute SignatureProductionPlaceV2 defined in clause 5.2.5 of ETSI EN 319 132-1 [30].
AllDataObjectsTimeStamp	This parameter contains the Base64-encoding of the attribute AllDataObjectsTimeStamp defined in clause 5.2.8.1 of ETSI EN 319 132-1 [30].
IndividualDataObjectsTimeStamp	This parameter contains the Base64-encoding of the attribute IndividualDataObjectsTimeStamp defined in clause 5.2.8.2 of ETSI EN 319 132-1 [30].
SignaturePolicyIdentifier	This parameter contains the Base64-encoding of the attribute SignaturePolicyIdentifier defined in clause 5.2.9 of ETSI EN 319 132-1 [30].

Output

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

Parameter	Presence	Value	Description
DocumentWithSignature	REQUIRED Conditional	*Array of String*	One or more Base64-encoded signatures enveloped within the documents. This element SHALL carry a value only if the client application requested the creation of signature(s) enveloped within the signed document(s) and when *operationMode* is not "A".
SignatureObject	REQUIRED Conditional	*Array of String*	One or more Base64-encoded signatures detached from the documents. This element SHALL carry a value only if the client application requested the creation of detached signature(s) and when *operationMode* is not "A".

Parameter	Presence	Value	Description
responseID	REQUIRED Conditional	*String*	The *responseID* as defined in the Output attribute table in signatures/signHash.
validationInfo	REQUIRED Conditional	*JSON Object*	The *validationInfo* is a JSON Object containing validation data that SHALL be included in the signing response if requested using the input parameter "returnValidationInfo".

The `validationInfo` is a JSON Object composed by the following parameters:

- `ocsp`
- `crl`
- `certificates`

specified according to the following table.

Parameter	Presence	Value	Description
ocsp	REQUIRED Conditional	*Array of String*	*ocsp* is an array of base64 encoded strings containing the DER-encoded ASN.1 data structures of type `OCSPResponse` according to RFC 6960 [33]. This value SHALL be included if at least one OCSP response is needed to validate the created signature and timestamps contained in the signature. It SHALL contain all needed OCSP responses. If for the same certificate an OCSP response and a CRL is available, the OCSP response SHOULD be included.
crl	REQUIRED Conditional	*Array of String*	*crl* is an array of base64 encoded strings containing the DER-encoded ASN.1 data structures of type `CertificateList` according to RFC 5280 [8]. This value SHALL be included if at least one CRL is needed to validate the created signature and timestamps contained in the signature. It SHALL contain all needed CRLs.
certificates	REQUIRED Conditional	*Array of String*	*certificates* contains one or more Base64-encoded X.509v3 certificates from the certificate chain used to create the respective signature and timestamps included in the signature. This value SHALL be included if at least one certificate is needed to validate the created signature and timestamps, which is not yet included in the signature. It SHALL contain all needed certificates.

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
When present, invalid object "documentDigests" parameter	400 (bad request)	invalid_request	Invalid object parameter documentDigests
When present, invalid array "documents" parameter	400 (bad request)	invalid_request	Invalid array parameter documents

Error Case	Status Code	Error	Error Description
Empty documentDigests and documents parameters	400 (bad request)	invalid_request	Empty documentDigests and documents objects
Both documentDigests and documents parameters have been passed	400 (bad request)	invalid_request	Both documentDigests and documents parameters passed
Invalid Base64 hashes element	400 (bad request)	invalid_request	Invalid Base64 hashes string parameter
Invalid Base64 documents element	400 (bad request)	invalid_request	Invalid Base64 documents string parameter
Unauthorized documentDigests or documents	400 (bad request)	invalid_request	documentDigests or documents are 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
“hashAlgorithmOID” parameter contradicting with “signAlgo” parameter	400 (bad request)	invalid_request	String parameter hashAlgorithmOID contradicts with signAlgo parameter
When present, 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 “signature_format” parameter	400 (bad request)	invalid_request	Invalid parameter signature_format
When “documents” is passed, missing or not String “signature_format” parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter signature_format
When present, invalid “conformance_level” parameter	400 (bad request)	invalid_request	Invalid parameter conformance_level
When present, invalid “signed_envelope_property” parameter	400 (bad request)	invalid_request	Invalid parameter signed_envelope_property
When present, invalid “signed_props” parameter	400 (bad request)	invalid_request	Invalid parameter signed_props (list of invalid attributes)
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

Error Case	Status Code	Error	Error Description
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” element length	400 (bad request)	invalid_request	Invalid digest value length
Expired “SAD”	400 (bad request)	invalid_request	SAD expired
Expired credential	400 (bad request)	invalid_request	Signing certificate ‘O=[organization],CN=[common_name]’ is expired.
Document or documentDigest to be signed does not match one of the authorized hashes	403 (bad request)	invalid_hash	Document or documentDigest does not match authroized hash

Sample Request

```
POST /csc/v2/signatures/signDoc 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",
    "documentDigests": [
        {
            "hashes": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
            "signature_format": "P",
            "conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "hashes": "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
            "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
            "signature_format": "C",
            "conformance_level": "AdES-B-B",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ],
    "documents": [
        {
            "document": "Q2VydGlmaWNhdGVTZXJpYWxOdW1iZ…KzBTWWVJWWZZVXptU3V5MVU9DQo=",
            "signature_format": "P",
            "conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "document": "Q2VydGlmaWNhdGVTZXJpYWxOdW1iZXI7U3… emNNbUNiL1cyQT09DQo=",
            "signature_format": "C",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Attached",
            "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",
    "documentDigests": [
        {
            "hashes": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
            "signature_format": "P",
            "conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "hashes": "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0=",
            "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
            "signature_format": "C",
            "conformance_level": "AdES-B-B",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ],
    "documents": [
        {
            "document": "Q2VydGlmaWNhdGVTZXJpYWxOdW1iZ…KzBTWWVJWWZZVXptU3V5MVU9DQo=",
            "signature_format": "P",
            "conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1"
        },
        {
            "document": "Q2VydGlmaWNhdGVTZXJpYWxOdW1iZXI7U3… emNNbUNiL1cyQT09DQo=",
            "signature_format": "C",
            "conformance_level": "AdES-B-B",
            "signed_envelope_property": "Attached",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ],
    "clientData": "12345678"
}'
https://service.domain.org/csc/v2/signatures/signDoc
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "DocumentWithSignature":
    [
        "MILuLgYJKoZIhvcNAQcCoILuHz… ehEeR5ZRi5+WV5T1FpO",
        "MIL4IAYJKoZIhvcNAQcCoIL4…YavvBxkVwJ3dFD9KbCi1qW3TxTI="
    ],
    "SignatureObject":
    [
        "MIAGCSqAMIACAQExDzANBglghkgBZQMEAgEFADCABgkqhkiG…Ss4rEsQV4AAAAAAAAA==",
        "MIAGCSqGSIb3DQEHAqCAMIACAQExDzANBglghkgBZQMEqhki…W7pP1ZJFKuF2YAAAAAAA"
    ]
}
```

Sample Request

```
POST /csc/v2/signatures/signDoc HTTP/1.1 Host: service.domain.org
Content-Type: application/json
Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA
{
    "signatureQualifier": "eu_eidas_qes",
    "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
        "documentDigests": [
        {
            "hashes": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
            "signature_format": "P",
            "conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1"
        }
    ],
    "clientData": "12345678",
    "returnValidationInfo":true
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{
        "signatureQualifier": "eu_eidas_qes",
    "SAD": "_TiHRG-bAH3XlFQZ3ndFhkXf9P24/CKN69L8gdSYp5_pw",
    "documentDigests": [
         {
             "hashes": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
             "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1",
             "signature_format": "P",
             "conformance_level": "AdES-B-T",
             "signAlgo": "1.2.840.113549.1.1.1"
          } ],
        "clientData": "12345678",
        "returnValidationInfo":true}'
https://service.domain.org/csc/v2/signatures/signDoc
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "SignatureObject":
    [
        "MIAGCSqAMIACAQExDzANBglghkgBZQMEAgEFADCABgkqhkiG…Ss4rEsQV4AAAAAAAAA==",
        "MIAGCSqGSIb3DQEHAqCAMIACAQExDzANBglghkgBZQMEqhki…W7pP1ZJFKuF2YAAAAAAA"
    ],
    "validationInfo":{
        "ocsp":[
           "MIIJg...jSc="
        ],
        "crl":[
           "MIIC4...X7M="
        ]
        "certificates":[
            "<Base64-encoded_X.509_certificate>"
        ]
    }
}
```

Sample Request

```
POST /csc/v2/signatures/signDoc HTTP/1.1 Host: service.domain.org
Content-Type: application/json
Authorization: Bearer 6/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA
{
    "signatureQualifier": "qes_eidas",
    "documentDigests":
    {
        "hashes":
        [
            "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
            "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0="
        ],
        "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
    },
    " signature_format": "P",
    " conformance_level": "AdES-B-T",
    "signAlgo": "1.2.840.113549.1.1.1",
    "clientData": "12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 6/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{  "signatureQualifier": "qes_eidas",
            "documentDigests":
            {
                "hashes":
                [
                    "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
                    "HZQzZmMAIWekfGH0/ZKW1nsdt0xg3H6bZYztgsMTLw0="
                ],
                "hashAlgorithmOID": "2.16.840.1.101.3.4.2.1"
            },
            " signature_format": "P",
            " conformance_level": "AdES-B-T",
            "signAlgo": "1.2.840.113549.1.1.1",
            "clientData": "12345678"}'
https://service.domain.org/csc/v2/signatures/signDoc
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "SignatureObject":
    [
        "MIAGCSqAMIACAQExDzANBglghkgBZQMEAgEFADCABgkqhkiG…Ss4rEsQV4AAAAAAAAA==",
        "MIAGCSqGSIb3DQEHAqCAMIACAQExDzANBglghkgBZQMEqhki…W7pP1ZJFKuF2YAAAAAAA"
    ]
}
```

11.12 signatures/signPolling

Description

Request to the server to return the responses corresponding to previously sent (initial) digital signature value(s) or signature(s) creation request when processed in asynchronous mode.

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

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
requestID	REQUIRED	*String*	The value generated by the server uniquely identifying the response originated from the server itself to a previous asynchronous signature request.
userID	REQUIRED Conditional	*String*	The *userID* as defined in the Input parameter table in credentials/list.
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:

Parameter	Presence	Value	Description
signatures	REQUIRED Conditional	*Array of String*	The *signatures* as defined in the Output attribute table in signatures/signHash. This element SHALL carry a value only if the client application requested the creation of digital signature value(s). This value SHALL be returned when the requested digital signature(s) creation has been completed.
DocumentWithSignature	REQUIRED Conditional	*Array of String*	The *DocumentWithSignature* as defined in the Output attribute table in signatures/signDoc. This value SHALL be returned when the requested signature(s) creation has been completed.
SignatureObject	REQUIRED Conditional	*Array of String*	The *SignatureObject* as defined in the Output attribute table in signatures/signDoc. This value SHALL be returned when the requested signature(s) creation has been completed.

Error Case	Status Code	Error	Error Description
The previous asynchronous signature request has been accepted for processing	202 (accepted)	accepted_request	The previous asynchronous signature request has been accepted for processing, but the processing has not yet been completed.
The authorization header does not match the pattern "Bearer [sessionKey]"	400 (bad request)	invalid_request	Malformed authorization header.
Missing or not String "requestID" Parameter	400 (bad request)	invalid_request	Missing (or invalid type) string parameter requestID
Invalid requestID parameter	400 (bad request)	invalid_request	Invalid parameter requestID
Not empty "userID" parameter in case of user- specific authorization	400 (bad request)	invalid_request	userID parameter SHALL be null

Error Case	Status Code	Error	Error Description
Invalid “userID” format in case of no user-specific authorization	400 (bad request)	invalid_request	Invalid parameter “userID”
When present, invalid “clientData” format (not string)	400 (bad request)	invalid_request	Invalid parameter clientData

Sample Request

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

{
   "requestID":"158112-652341-khj",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "requestID":"158112-652341-khj",
           "clientData": "12345678" }'
     https://service.domain.org/csc/v2/signatures/signPolling
```

Sample Response

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

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

}
```

11.13 signatures/timestamp

Description

Generate a time-stamp token for the input hash value. The time-stamp token can be generated directly by the RSSP or by a Time Stamping Authority connected to it.

The reason to implement this method instead of providing time-stamp services through widespread RFC 3161 [2] protocols directly is to facilitate the creation of long-term validation digital signatures and to support billing operations. In both cases, the RSSP provider can offer pre-configured time-stamp services instead of requiring the signature application to obtain time-stamp services from a different provider.

Input

This method allows the following parameters:

Parameter	Presence	Value	Description
hash	REQUIRED	*String*	The Base64-encoded hash value to be time stamped. The remote service SHALL use this value to encode the value of MessageImprint.hashedMessage as defined in RFC 3161 [2].
hashAlgo	REQUIRED	*String*	The OID of the algorithm used to calculate the hash value. The remote service SHALL use this value to encode the value of MessageImprint.hashAlgorithm as defined in RFC 3161 [2].
nonce	OPTIONAL	*String*	A large random number with a high probability that it is generated by the signature application only once. The value SHALL be represented as hex-encoded string.
clientData	OPTIONAL	*String*	The *clientData* as defined in the Input parameter table in oauth2/authorize.

Note 34: RFC 3161 [2] contains more detailed definitions of time stamp parameters that can be used in the context of this specification.

Output

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

Parameter	Presence	Value	Description
timestamp	REQUIRED	*String*	The Base64-encoded time-stamp token as defined in RFC 3161 [2] as updated by RFC 5816 [10]. If the *nonce* parameter is included in the request then it SHALL also be included in the time-stamp token, otherwise the response SHALL be rejected.

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.
The"hash" parameter is missing or not of type String.	400 (bad request)	invalid_request	Missing (or invalid type) string parameter hash
Empty hash parameter	400 (bad request)	invalid_request	Empty hash parameter
Invalid "hash" length	400 (bad request)	invalid_request	Invalid digest value length
Invalid Base64 hash element	400 (bad request)	invalid_request	Invalid Base64 hash string parameter
Invalid "hashAlgo" parameter	400 (bad request)	invalid_request	Invalid parameter hashAlgo
Invalid or non-numeric "nonce" parameter	400 (bad request)	invalid_request	Invalid parameter nonce

Sample Request

```
POST /csc/v2/signatures/timestamp HTTP/1.1
Host: service.domain.org
```

```
Content-Type: application/json
Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA

{
   "hash":"sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
   "hashAlgo":"2.16.840.1.101.3.4.2.1",
   "clientData":"12345678"
}
```

cURL example

```
curl -X POST
     -H "Content-Type: application/json"
     -H "Authorization: Bearer 4/CKN69L8gdSYp5_pwH3XlFQZ3ndFhkXf9P2_TiHRG-bA"
     -d '{ "hash": "sTOgwOm+474gFj0q0x1iSNspKqbcse4IeiqlDg/HWuI=",
           "hashAlgo": "2.16.840.1.101.3.4.2.1",
           "clientData": "12345678" }'
     https://service.domain.org/csc/v2/signatures/timestamp
```

Sample Response

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

{

"timestamp":"MGwCAQEGCSsGAQQB7U8CATAxMA0GCWCGSAFlAwQCAQUABCCrCqnrjH0VxXyQQlfnFJRx1jjrviTs
7/GjKghr2AmluQIIVs5D8OUB4p4YDzIwMTQxMTE5MTEzMjM5WjADAgEBAgkAnWn2SSIWlXk="
}
```

12 JSON schema and OpenAPI description

A signature application may want to validate the JSON objects described in this specification, to ensure that required properties are present and that additional constraints are met. Validation of JSON data is typically performed by means of a specific JSON Schema.

A JSON Schema is a grammar language for defining the structure, content, and semantics of JSON data objects. It can specify metadata about the meaning of an object's properties and values that are valid for those properties. The JSON Schema is defined at https://json-schema.org.

The JSON schema of the API specification described in this specification is available from the website of the Cloud Signature Consortium at:
https://cloudsignatureconsortium.org/resources/download-api-specifications/.

The JSON Schema file contains the definition of all CSC API parameters and the definition of the input and output objects managed by the CSC API. The following objects are defined:

- *input-info:* input object for info method
- *output-info:* output object for info method
- *input-auth-login:* input object for auth/login method
- *output-auth-login:* output object for auth/login method
- *input-auth-revoke:* input object for auth/revoke method

- *input-credentials-list:* input object for credentials/list method
- *output-credentials-list:* output object for credentials/list method
- *input-credentials-info:* input object for credentials/info method
- *output-credentials-info:* output object for credentials/info method
- *input-credentials-authorize:* input object for credentials/authorize method
- *output-credentials-authorize:* output object for credentials/authorize method
- *input-credentials-extendTransaction:* input object for credentials/extendTransaction method
- *output-credentials-extendTransaction:* output object for credentials/extendTransaction method
- *input-credentials-sendOTP:* input object for credentials/sendOTP method
- *input-signatures-signhash:* input object for signatures/signhash method
- *output-signatures-signhash:* output object for signatures/signhash method
- *input-signatures-timestamp:* input object for signatures/timestamp method
- *output-signatures-timestamp:* output object for signatures/timestamp method

In addition, an OpenAPI 3.0 description file is provided, as defined by the OpenAPI Initiative (OAI) https://www.openapis.org, containing these JSON Schema definitions together with other information to fully describe the CSC API protocol. The OpenAPI file contains:

1. A general information about the protocol like, for example, the APIs version, the Cloud Signature Consortium contact information and the license;
2. Information about the RESTful path URL and an example of server URL access points;
3. Authorization schemas required to access the CSC API;
4. A description of every method of the CSC protocol including input objects and returned HTTP responses.

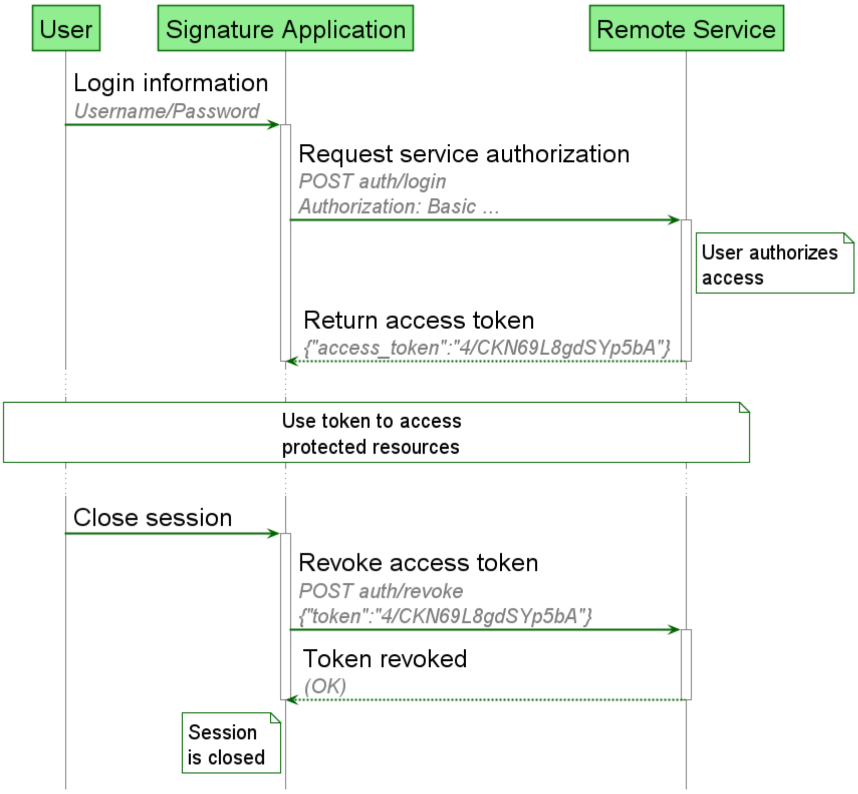
The OpenAPI description file can also be used by developers or testers to automatically generate a CSC compliant server interfaces or client stubs.

13 Interaction among elements and components

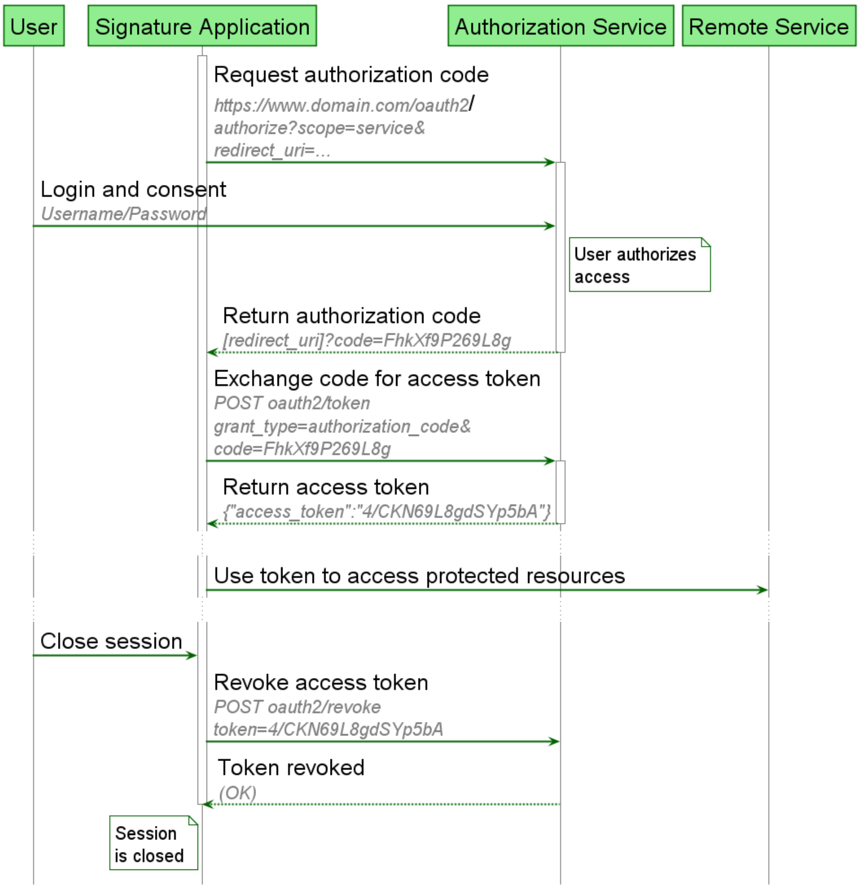
The building blocks of a remote signature solution interact with the API methods described in this specification. The following sections describe the sequence diagrams of some of the most common operations required to obtain a service authorization, credential authorization and to request a remote signature.

Note 35: The sample requests and responses that are provided in the diagrams are only a partial representation of complete transactions and are aimed at showing the most important parameters and information. See the example in the previous sections of this specification for complete and detailed descriptions.

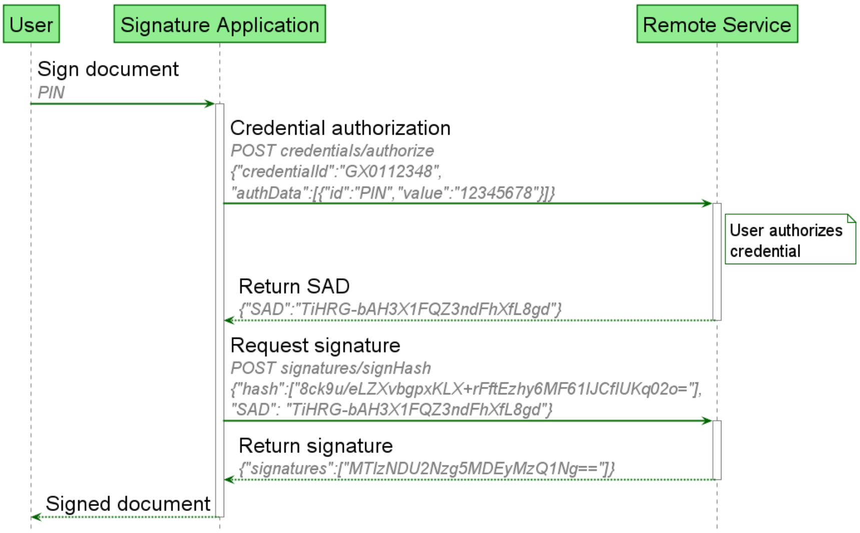
13.1 Remote signing service authorization using Basic Authentication



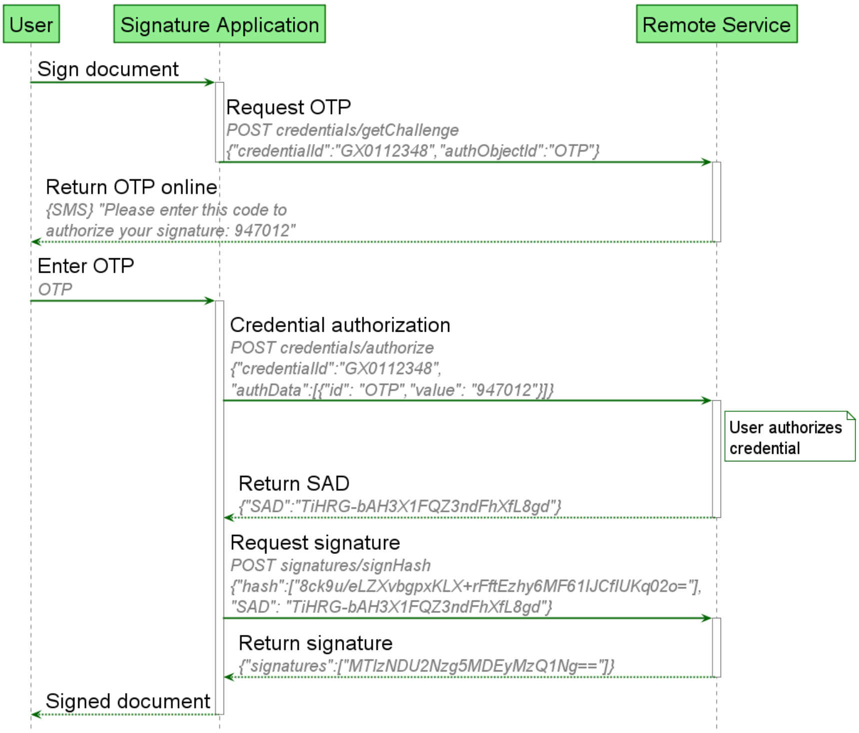
13.2 Remote signing service authorization using OAuth2 with Authorization Code flow



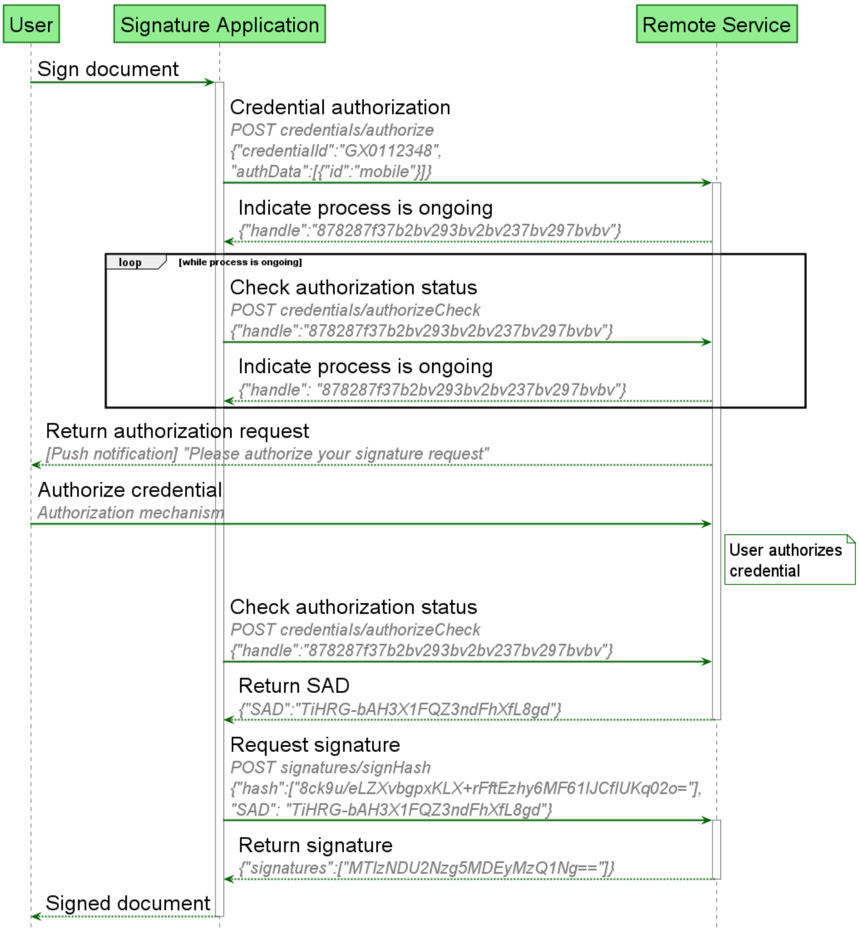
13.3 Create a remote signature with a credential protected by a PIN


User
Signature Application
Remote Service
Sign document
PIN
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348",
"authData":[{"id":"PIN","value":"12345678"}]}
User authorizes
credential
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61lJCfIUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

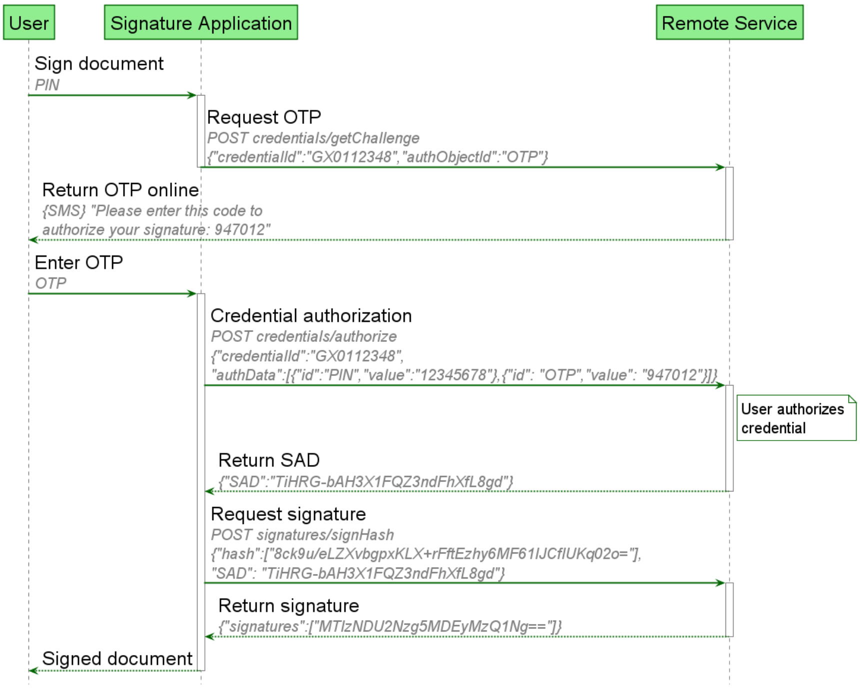
13.4 Create a remote signature with a credential protected by an "online" OTP (based on SMS)


User
Signature Application
Remote Service
Sign document
Request OTP
POST credentials/getChallenge
{"credentialId":"GX0112348","authObjectId":"OTP"}
Return OTP online
{SMS} "Please enter this code to
authorize your signature: 947012"
Enter OTP
OTP
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348",
"authData":[{"id": "OTP","value": "947012"}]}
User authorizes
credential
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61IJCfIUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

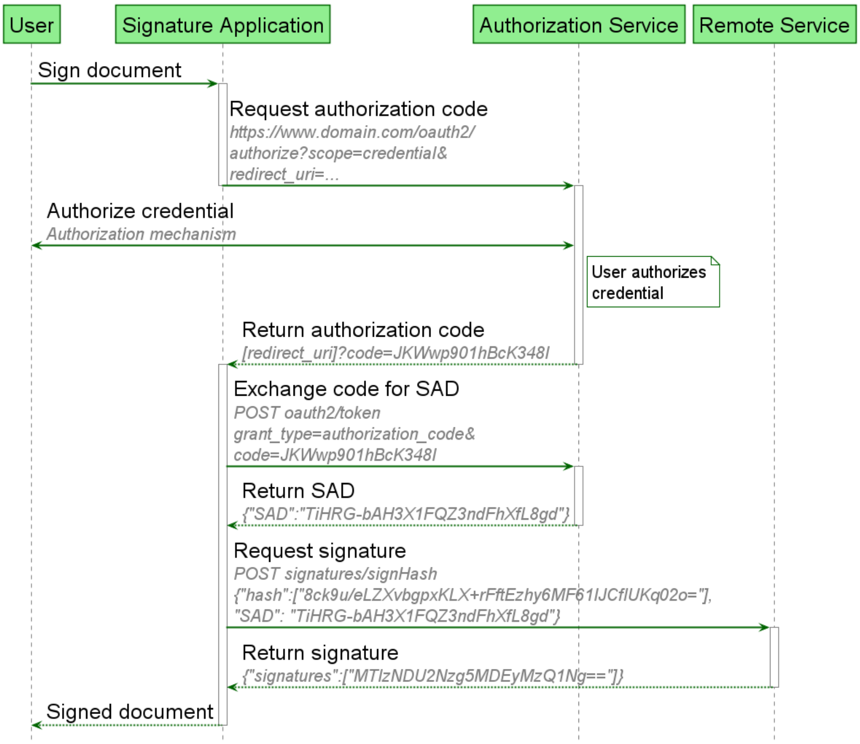
13.5 Create a remote signature with a credential protected by a mobile app


User
Signature Application
Remote Service
Sign document
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348",
"authData":[{"id":"mobile"}]}
Indicate process is ongoing
{"handle":"878287f37b2bv293bv2bv237bv297bvbv"}
loop
[while process is ongoing]
Check authorization status
POST credentials/authorizeCheck
{"handle":"878287f37b2bv293bv2bv237bv297bvbv"}
Indicate process is ongoing
{"handle": "878287f37b2bv293bv2bv237bv297bvbv"}
Return authorization request
[Push notification] "Please authorize your signature request"
Authorize credential
Authorization mechanism
User authorizes
credential
Check authorization status
POST credentials/authorizeCheck
{"handle":"878287f37b2bv293bv2bv237bv297bvbv"}
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61IJCfIUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

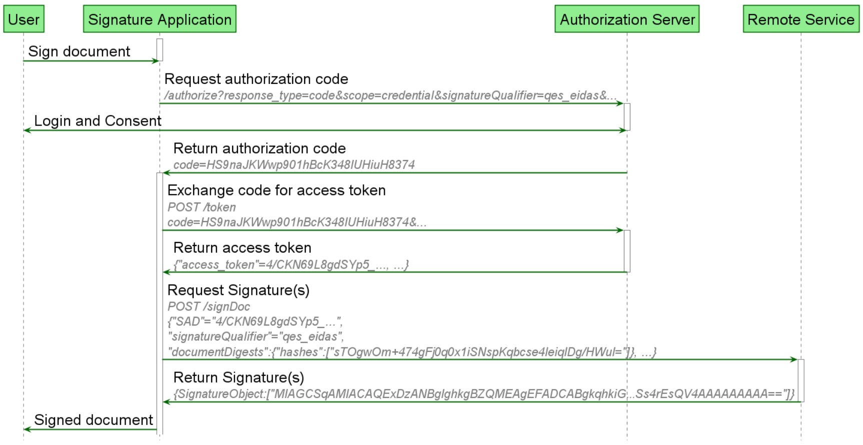
13.6 Create a remote signature with a credential protected by a PIN and an “online” OTP (based on SMS)


User
Signature Application
Remote Service
Sign document
PIN
Request OTP
POST credentials/getChallenge
{"credentialId":"GX0112348","authObjectId":"OTP"}
Return OTP online
{SMS} "Please enter this code to
authorize your signature: 947012"
Enter OTP
OTP
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348",
"authData":[{"id":"PIN","value":"12345678"},{"id": "OTP","value": "947012"}]}
User authorizes
credential
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61lJCflUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

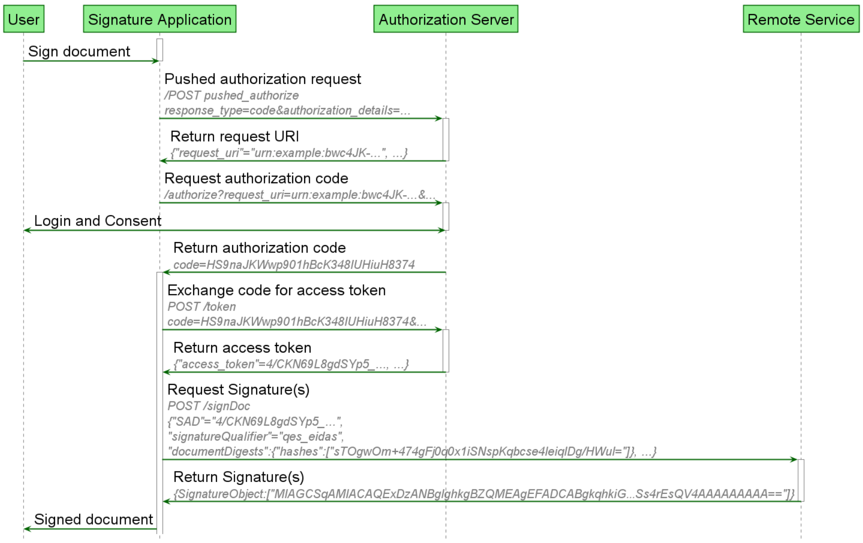
13.7 Create a remote signature with a credential protected by OAuth2 with Authorization Code flow


User
Signature Application
Authorization Service
Remote Service
Sign document
Request authorization code
https://www.domain.com/oauth2/
authorize?scope=credential&
redirect_uri=...
Authorize credential
Authorization mechanism
User authorizes
credential
Return authorization code
[redirect_uri]?code=JKWwp901hBcK348l
Exchange code for SAD
POST oauth2/token
grant_type=authorization_code&
code=JKWwp901hBcK348l
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61lJCflUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

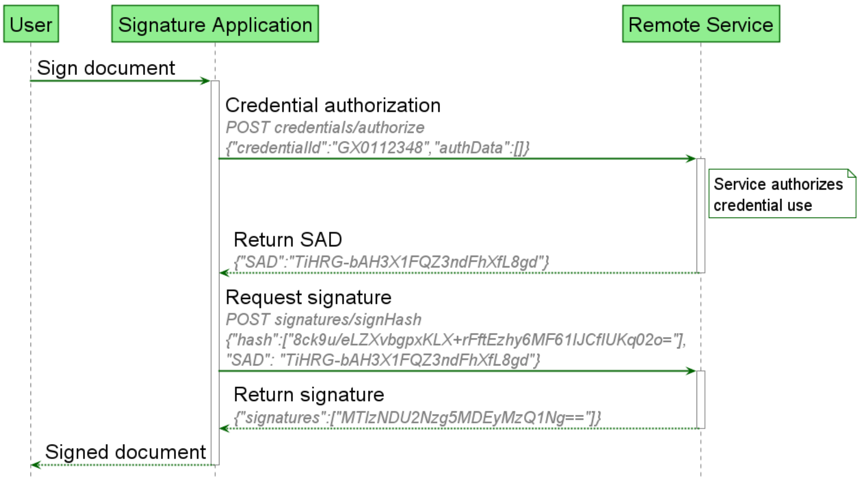
13.8 Create a remote signature with credential and signature qualifier with OAuth2 Authorization Code flow


User
Signature Application
Authorization Server
Remote Service
Sign document
Request authorization code
/authorize?response_type=code&scope=credential&signatureQualifier=qes_eidas&...
Login and Consent
Return authorization code
code=HS9naJKWwp901hBcK348IUHiuH8374
Exchange code for access token
POST /token
code=HS9naJKWwp901hBcK348IUHiuH8374&...
Return access token
{"access_token"=4/CKN69L8gdSYp5_..., ...}
Request Signature(s)
POST /signDoc
{"SAD"="4/CKN69L8gdSYp5_...",
"signatureQualifier"="qes_eidas",
"documentDigests":{"hashes":["sTOgwOm+474gFj0q0x1iSNspKqbcse4leiqlDg/HWul="]}, ...}
Return Signature(s)
{SignatureObject:["MIAGCSqAMIACAQExDzANBglghkgBZQMEAgEFADCABgkqhkiG...Ss4rEsQV4AAAAAAAAA=="]}
Signed document

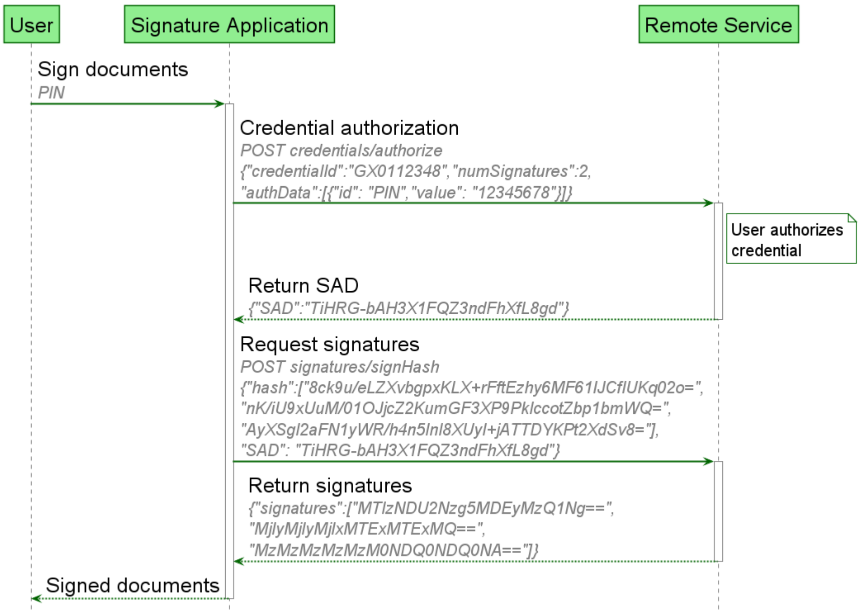
13.9 Create a remote signature with OAuth2 Authorization Code flow and Pushed and Rich Authorization Request


User
Signature Application
Authorization Server
Remote Service
Sign document
Pushed authorization request
/POST pushed_authorize
response_type=code&authorization_details=...
Return request URI
{"request_uri"="urn:example:bwc4JK-...", ...}
Request authorization code
/authorize?request_uri=urn:example:bwc4JK-...&...
Login and Consent
Return authorization code
code=HS9naJKWwp901hBcK348IUHiuH8374
Exchange code for access token
POST /token
code=HS9naJKWwp901hBcK348IUHiuH8374&...
Return access token
{"access_token"=4/CKN69L8gdSYp5_..., ...}
Request Signature(s)
POST /signDoc
{"SAD"="4/CKN69L8gdSYp5_...",
"signatureQualifier"="qes_eidas",
"documentDigests":{"hashes":["sTOgwOm+474gFj0q0x1iSNspKqbcse4leiqlDg/HWul="]}, ...}
Return Signature(s)
{SignatureObject:["MIAGCSqAMIACAQExDzANBglghkgBZQMEAgEFADCABgkqhkiG...Ss4rEsQV4AAAAAAAAA=="]}
Signed document

13.10 Create a remote signature with a credential protected by RSSP-managed authorization

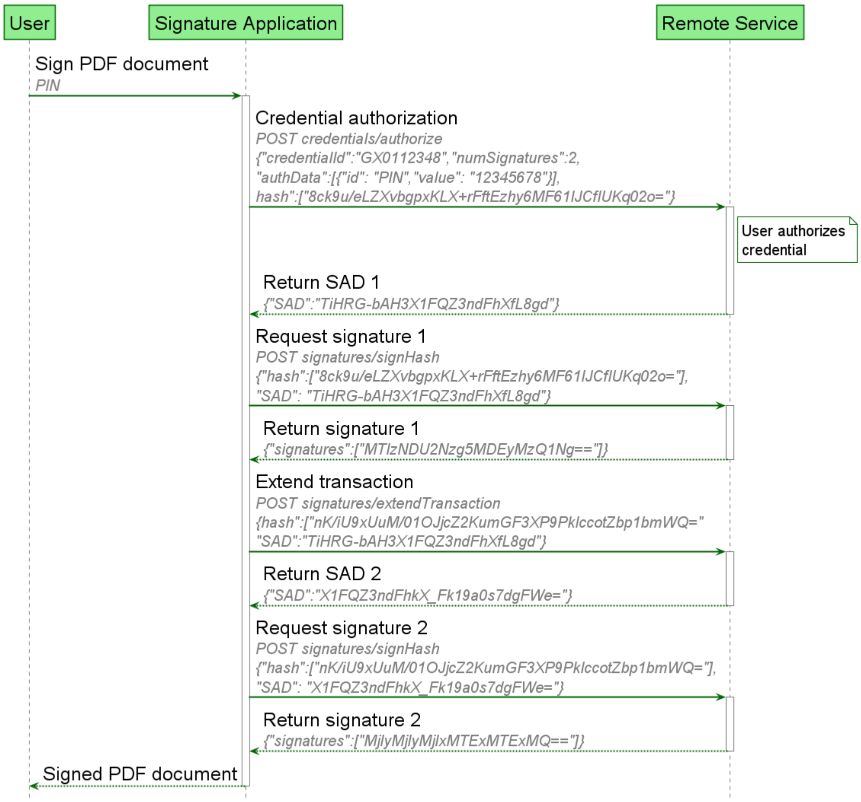

User
Signature Application
Remote Service
Sign document
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348","authData":[]}
Service authorizes credential use
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61lJCflUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Signed document

13.11 Create multiple remote signatures from a list of hash values


User
Signature Application
Remote Service
Sign documents
PIN
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348","numSignatures":2,
"authData":[{"id": "PIN","value": "12345678"}]}
User authorizes credential
Return SAD
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signatures
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61lJCflUKq02o=",
"nK/iU9xUuM/01OJjcZ2KumGF3XP9PklccotZbp1bmWQ=",
"AyXSgl2aFN1yWR/h4n5lnl8XUyl+jATTDYKPt2XdSv8="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signatures
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng==",
"MjIyMjIyMjIxMTExMTExMQ==",
"MzMzMzMzM0NDQ0NDQ0NA=="]}
Signed documents

13.12 Create a remote multi-signatures transaction with a PDF document

This diagram shows the case of a PDF document that is signed multiple times by the same signer. A single credential authorization can be performed to authorize multiple signatures. However only the initial hash of the document is available at authorization time. A new hash will be generated to calculate the following signatures. For this reason, the **credentials/extendTransaction** method is used to supply the new hash to obtain the SAD to calculate a new signature. See credentials/extendTransaction for more information.


User
Signature Application
Remote Service
Sign PDF document
PIN
Credential authorization
POST credentials/authorize
{"credentialId":"GX0112348","numSignatures":2,
"authData":[{"id": "PIN","value": "12345678"}],
hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61IJCfIUKq02o="}
User authorizes credential
Return SAD 1
{"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Request signature 1
POST signatures/signHash
{"hash":["8ck9u/eLZXvbgpxKLX+rFftEzhy6MF61IJCfIUKq02o="],
"SAD": "TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return signature 1
{"signatures":["MTIzNDU2Nzg5MDEyMzQ1Ng=="]}
Extend transaction
POST signatures/extendTransaction
{hash":["nK/iU9xUuM/01OJjcZ2KumGF3XP9PklccotZbp1bmWQ="
"SAD":"TiHRG-bAH3X1FQZ3ndFhXfL8gd"}
Return SAD 2
{"SAD":"X1FQZ3ndFhkX_Fk19a0s7dgFWe="}
Request signature 2
POST signatures/signHash
{"hash":["nK/iU9xUuM/01OJjcZ2KumGF3XP9PklccotZbp1bmWQ="],
"SAD": "X1FQZ3ndFhkX_Fk19a0s7dgFWe="}
Return signature 2
{"signatures":["MjlyMjlyMjlxMTExMTExMQ=="]}
Signed PDF document

CLOUD
SIGNATURE
CONSORTIUM

14 Change history

14.1 Changes since version 1.0.4.0

- Add certificate info into credentials/list method: It is now allowed to provide directly in the credentials/list method the detailed information of the certificates.

- Add asymmetric signing: The possibility was added to use asynchronous call as was already proposed in ETSI TS 119 432.

- Add signing of documents: It is not only possible to create a cryptographic signature of a hash, but also to create an AdES signature on a hash or a document. The functionality is more powerful than the one introduced in ETSI TS 119 432 since it allows different signature formats for different documents within one call, which is especially useful in case a certificate is only created for one signature authorization, and this authorization should cover different types of documents. It also allows to request JAdES signatures.

- Possibility to use authorization request payload (PAR) and rich authorization requests (RAR) in the OAuth authorization.

- Allow to use only the credential OAuth authorization (without service authorization) for signing.

- Add a chapter on the usage of the CSC protocol for creating electronic seals.

- When creating a PAdES signature based on the hash document, provide the revocation information so that this can be included in the final signed document.

- Allow to request only credential which are valid, i.e. which can be used to sign, in the credentials/list endpoint

- Add explanation how to define algorithms via OIDs.

- Allow to request signature authorization via OAuth on a credential of a specific type, without specifying the credential ID. This is useful for short lived credentials which are only created for a specific signature process.

- Make explicit credential authorization more flexible: The explicit credential authorization allows to use and combine different authorization types. This makes the implicit credential authorization useless, because it can be expressed as part of the explicit credential authorization.

- Each time hash values are provided, provide also the hash algorithm