DIGITAL AND POPULATION DATA SERVICES AGENCY

HTML5 and Digital Signatures

Signature Creation Service 1.2

16.10.2023

Tel. **+358 295 536 000** (exchange) | Contact Information, see **www.dvv.fi/en**

Document management

Owner	
Prepared by	Pekka Laitinen / VRK Antti Partanen / VRK Jari Pirinen / DVV jari.pirinen@dvv.fi Seppo Pekkala / Fujitsu Finland seppo.pekkala@fujitsu.com
Inspected by	
Approved by	

Version control

version no.	what has been done	date/person
1.0	Final version (no changes to 0.7 version)	17.2.2015/PL
1.0.1	Added: • https usage as requirement • server certificate requirements • SCS module for javascript (example) • clarifications for selector.akis and selector.issuers format • removed selector.validate parameter • added requirement for SCS to check whether certificate is valid or not • removed localhost.fineid.fi as dnsName parameter in SCS server certificate (compared to previous 1.0.1 version) protocol version is still the same: 1.0	30.6.2015/PL
1.1	Added: • support for http based access dropped completely • separation of authentication and signing purpose • authentication challenge to be used for authentication purpose • keyalgorithms array to selector functionality • cms signature type and cms signature profile • protocol version 1.1 • support signature request size up to 100MB (was 2MB) minor editorial corrections	4.6.2017/PL 21.11.2017/AP
1.2	Moved to DVV document template Added: • “cms-pades” and “signature-plain” signature types • RSASSA-PSS support • transaction based signing Removed: • SHA1 support	1.3.2022/JP 1.3.2022/SP
1.2.1	Transaction based signature: • Refined server, browser and SCS signer concepts • Algorithm identifier correction (EC256 -> ES256) • Added a diagram of the steps of the signing process	16.10.2023/SP

Kehitys ja tietohallinto / Jari Pirinen 16.10.2023

Kehitys ja tietohallinto / Jari Pirinen

16.10.2023

Table of contents

1 Introduction 5
1.1 Definitions and Acronyms 5
1.1 References 6
2 Signature Creation Service (SCS) 7
2.1 Requirements 7
2.2 Message sequence diagram 10
2.3 Server certificate 12
2.4 CORS preflight check 13
2.5 Version check 14
2.5.1 Request 14
2.5.2 Response 14
2.6 Signature creation 17
2.6.1 Request (POST) 17
2.6.2 Request (GET) 20
2.6.3 Response 20
2.7 Transactions 23
2.7.1 Begin transaction 23
2.7.2 Execute transaction 28
2.8 Reason codes 32
3 Security considerations 34
3.1 Authentication 34
3.2 Confidentiality, Integrity, Privacy 34
3.3 Local server socket 34
3.4 End user permission 34
3.5 Identifying origin 34
3.6 Non-repudiation signatures 34
3.7 Nonce 35
3.8 Certificate types 35
3.9 Malware 35
3.10 Multi-user environments 36
3.11 Browser policies 36
4 SCS Profiles 37
4.1 Base profile 37
4.2 RSA signature profile 37
4.3 ECDSA profile 38

4.4 CMS-pades profile 39

HTML5 and Digital Signatures

1 Introduction

This specification describes a method to generate digital signatures in HTML5 applications [HTML5] that are executed in User Agents, i.e., web browsers. The specification utilizes the Cross-Origin Resource Sharing (CORS) specification [CORS] that enables an HTML5 application downloaded from Site A to communicate with a service located in Site B using Javascript's XMLHttpRequest mechanisms [XHR], for instance. The communication protocol uses HTTP protocol and the information elements are transferred using JSON format [JSON].

The HTML5 application makes a signature request by sending the data that needs to be signed to the Signature Creation Service (SCS). Upon receiving the request, the SCS displays a certificate selection dialog to the end user, who will select the certificate that will be used to generate the digital signature. If required, the end user enters the PIN code for accessing the private key to generate the signature. Once the signature is created, the SCS sends the signature along with the certificate chain and other needed information to the HTML5 application. Upon receiving the digital signature, the HTML5 application uses it according to its specifications.

1.1 Definitions and Acronyms

CMS	Cryptographic Message Syntax
CORS	Cross-Origin Resource Sharing
CRL	Certificate Revocation List
JSON	JavaScript Object Notation
OCSP	Online Certificate Status Protocol
OS	Operating System
PKCS	Public-Key Cryptography Standards
SCS	Signature Creation Service
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography

1.1 References

[CMS] Cryptographic Message Syntax (CMS), September 2009, http://www.ietf.org/rfc/rfc5652.txt

[PADES] Electronic Signatures and Infrastructures, ETSI EN 319 142-1 and EN 319 142-2, https://www.etsi.org/standards

[CORS] Cross-Origin Resource Sharing, W3C Recommendation, January 16, 2014, http://www.w3.org/TR/cors/

[ECDSA] Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), August 2013, http://www.ietf.org/rfc/rfc6979.txt

[X9.62] American National Standards Institute, Inc. (ANSI), Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA), November 2005

[SEC1] Standards for Efficient Cryptography (SEC), SEC 1: Elliptic Curve Cryptography, Certicom Research, May 21, 2009, Version 2.0

[HTML5] HTML5, W3C Candidate Recommendation, July 31, 2014, http://www.w3.org/TR/html5/

[HTTP] Hypertext Transfer Protocol -- HTTP/1.1, June 1999, https://www.ietf.org/rfc/rfc2616.txt

[JSON] JavaScript Object Notation Introduction, http://json.org

[JWT] Json Web Tokens; rfc7515 rfc7516 and rfc7519.

[XHR] XMLHttpRequest Level 1, W3C Working Draft, January 30, 2014, http://www.w3.org/TR/XMLHttpRequest/

[PKCS1] Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, February 2003, http://www.ietf.org/rfc/rfc3447.txt

[PKIX] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008 https://www.ietf.org/rfc/rfc5280.txt

[HTTPS] HTTP over TLS, May 2000, https://www.ietf.org/rfc/rfc2818.txt

2 Signature Creation Service (SCS)

The SCS functions as a simple web server that provides access to signature creation functionality via HTTP/1.1 protocol [HTTP].

2.1 Requirements

The SCS SHALL support the following functionality:

- The SCS SHALL support https scheme and it MAY support http scheme for browser access. The SCS SHALL generate a key pair, generate a server certificate and import that certificate to device's trusted certificate store. Additional requirements can be found in chapter 2.3.
- The SCS SHALL support CORS [CORS]:
  - Each HTTP response SHALL contain the `Access-Control-Allow-Origin` header with value "*".
  - When User Agent makes the CORS preflight request, i.e., the OPTIONS request, the corresponding HTTP response SHALL contain the following headers: `Access-Control-Max-Age` header with default value "`3600`", `Access-Control-Allow-Methods` header with default value "`GET, POST`", and `Access-Control-Allow-Headers` headers with default value "`Content-Type, Accept`".
- The SCS SHALL process all HTTP requests.
- The SCS SHALL show certificate selection dialog for every signature creation request it receives.
- The SCS SHALL process HTTP requests where the `Origin` header containing `https` protocol in the URL. HTTP requests with Origin header containing `http` protocol in the URL SHALL not be processed.
- The SCS SHALL show the content of the `Origin` header of the signature creation request to the end user (so that end user is able to identify the origin of the signature creation request). It MAY be included in the certificate selection dialog.
- The SCS SHALL have only one active signature creation request active at a time. If another signature creation request is received, the SCS shall silently ignore the request and send a response to the requesting HTML5 application that another request is already active.
- The SCS SHALL respond to all incoming requests.
- The SCS SHALL be able to handle HTTP requests where content length is up to 100MB (=100*1024*1024 bytes). The SCS MAY be able to support larger HTTP requests.

- The SCS SHALL be able categorize signature creation requests to be either in signing purpose category or authentication purpose category. This categorization SHALL be based on the capabilities of the end entity certificate, namely by checking the KeyUsage and ExtendedKeyUsage extensions of the certificate. If the end entity certificate can be used for authentication purposes, the SCS SHALL allow only operations that are allowed for authentication purposes.
    - An end entity certificate can be used for authentication purposes, if the KeyUsage extension has the DigitalSignature bit set, or the ExtendedKeyUsage extension has the ClientAuthentication OID present or the ServerAuthentication OID present.
    - An end entity certificate can be used for signing purposes, if the KeyUsage extension has the NonRepudiation bit set. In particular, the end entity certificate SHALL NOT have the authentication purposes key usages as specified in previous paragraph.

  NOTE: With FINEID profile, the end entity certificates that have the DigitalSignature bit set in the KeyUsage extension can only be used for authentication purposes. The end entity certificates that have the NonRepudiation bit set in the KeyUsage extension can be used for signing purposes.

- The client application SHALL be able to send the data that is to be signed for signing purposes, i.e., the SCS will receive the full to-be-signed data, and calculate the digest of this data itself.

- The client application SHALL be able to send a challenge request to be signed for authentication purposes. The challenge request SHALL be in the following format:

  challenge_request = origin | nonce

  where

  origin is the address of the web server from where the HTML5 application application was downloaded from, i.e., origin SHALL be the same as the content of the CORS's Origin header (e.g., "https://dvv.fineid.fi"), and

  nonce is an octet string that SHALL be random and at least 64 octets in length.

  The maximum length of the challenge_request SHALL be 1024 octets in length.

  The SCS SHALL validate that the challenge request is in correct format.

- The client application SHALL be able to send the digest of the data that is to be signed for signing, i.e., the client application will calculate the digest of the to-be-signed data, and send it to the SCS.

- The client application SHALL NOT send the challenge request in digest format for authentication purposes. The SCS SHALL not accept signing requests in the digest format.

- The SCS MAY validate each end entity certificate (that is available on the device) and its certificate chain. If SCS validates end entity certificates and finds out that one expired, this end entity certificate is not shown to the end user as a selection option.

  NOTE: Even if the SCS validates end entity certificates, the server consuming digital signatures generated by SCS must also validate the end entity certificate and its corresponding certificate chain when checking the signature.

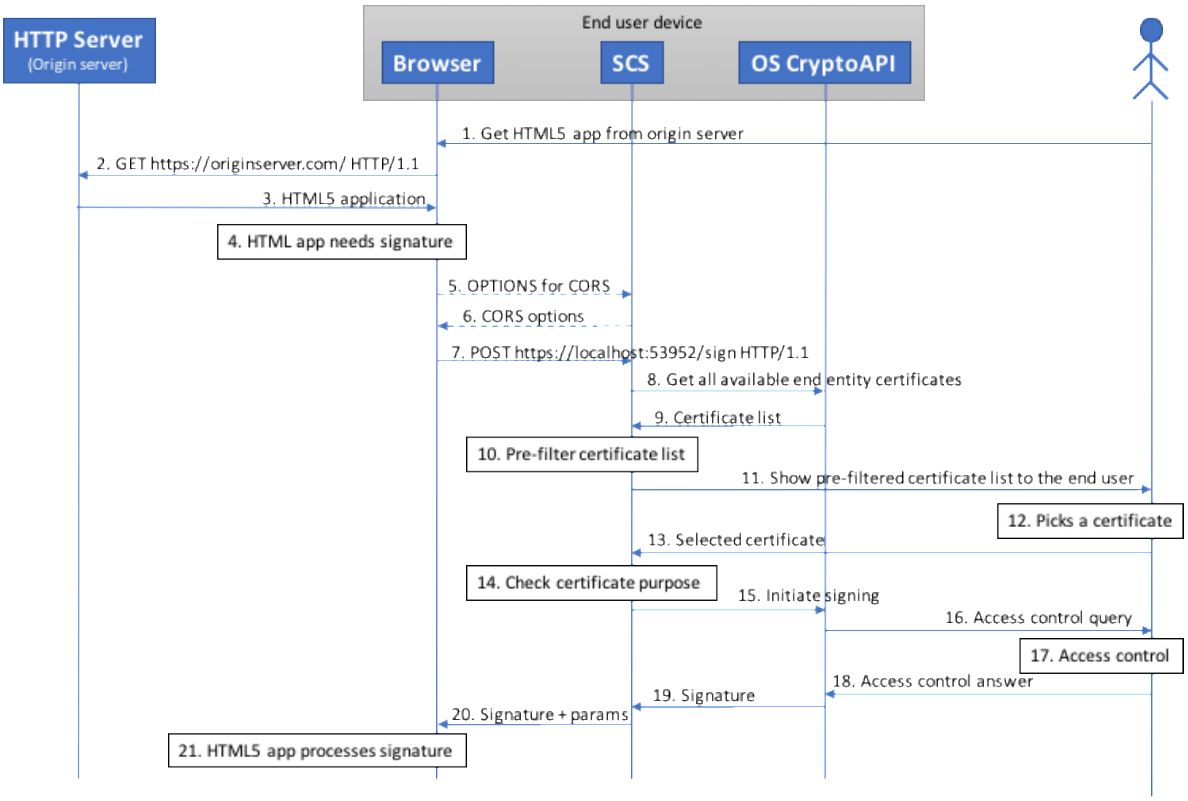
- The SCS MAY pre-filter applicable certificate list (before they are shown to the end user) and certificate filtering MAY be based on one or more values in end user certificates:

  - Key algorithm type,
  - Issuer DN field,
  - KeyUsage extension values, and/or
  - AuthorityKeyIdentifier extension.

  This way the HTML5 application is able to indicate to the SCS the acceptable end user certificates.

2.2 Message sequence diagram

Below is an example message sequence diagram of signature generation for HTML5 applications and SCS functionality. This specification concentrates on the interface between the browser and the SCS, namely steps 5, 6, 7, and 20. Moreover, the steps 5 and 6 are specified by the CORS specification. Steps from 8 to 19 are described for completeness but the actual steps depend on the OS architecture.



1. The end user types the URL of the origin server to the browser.

2. The browser makes a request to the URL. It is recommended that the URL uses https scheme.

3. The browser downloads the HTML5 application, which typically consists of several HTTP requests being made by the browser to the origin server. If the HTML5 application is downloaded using secure connection, i.e., https scheme, the HTTP requests to the SCS in step 5 and step 7 must be done over https scheme as well. If the HTML5 application is downloaded using unsecure connection, i.e., plain http scheme, the requests to the SCS in step 5 and step 7 must be done over http scheme as well.

4. At one point, application logic of the HTML5 application requires that a digital signature is needed.

5. The browser makes an OPTIONS request to the SCS to discover whether CORS type requests are allowed. This request may have been done also earlier.

6. The SCS returns the CORS options allowing requests being made from the current HTML5 application context (identified by Origin).

7. The HTML5 application makes a signature creation request.

8. The SCS queries local crypto API for all available end entity certificates.

9. The Crypto API returns all available end entity certificates.

10. The SCS pre-filters the end entity certificate list if signature creation request contained filters (selectors based on issuer name, authority key identifier or required key usage parameters). Additionally, the SCS may locally validate the end entity certificates, and filter out the ones that are not valid.

11. The SCS presents the end user with a list of (pre-filtered) end entity certificate list.

12. The end user selects one of the listed certificates.

13. The selected certificate is returned to the SCS.

14. The SCS checks the selected certificate whether it belongs to the signing purposes category or to the authentication purposes category. If the certificate belongs to the authentication purposes category, the signature creation request must in the form of challenge request. If it is not, the SCS will show a notification the end user that the signature creation request is not acceptable, and stops processing the request.

15. The SCS initiates the digital signature generation with the private key associated with the selected certificate and with given signature parameters received in the signature creation request.

16. Crypto API requests end user to give authorization to the usage of the selected private key.

17. End user enters the authorization data (e.g., PIN code).

18. Authorization data is returned to the Crypto API. If it was correct, the crypto API generates the signature.

19. Signature is returned to the SCS.

20. The SCS generates the signature creation response (with the signature, signature algorithm, and the certificate chain of the selected end entity certificate) and returns it to the HTML5 application.

21. The HTML5 application processes the signature including validating the certificate chain of the used end entity certificate (e.g., validity and revocation

status). The validation can also be done by the origin server (not depicted here).

Variations possibilities:

- After step 4, the HTML5 application may scan through the https ports that are listed in Base profile (section 4.1), and automatically detect the port that is being used by the SCS.
- Before showing the end user the certificate list for certificate selection, the SCS could show a notification what is being signed, i.e., show the actual data that is being signed. However, if data is not textual (e.g., XML signature element, or plain binary data), the end user might get confused by this notification.

2.3 Server certificate

In order for SCS to use https based scheme to communicate with browsers, it must generate a local key pair and a server certificate that needs to be imported to the devices trusted certificate store. The browser that wishes to use SCS must then use that trusted certificate store when validating server certificates.

Upon installation of the SCS to the target device, it SHALL generate two key pairs, one for the local root certificate that is going to be used sign the server certificate. After the certificates have been generated, the SCS SHALL delete the key pair of the root certificate, and store the key pair of the server certificate securely. The certificate chain, i.e., the root certificate and the server certificate SHALL be imported to the trusted certificate store of the target environment, or any other certificate store that the web browsers in the device are using.

The key pairs and the certificates have the following requirements:

- The key pairs SHALL be at least 2048 bits in the case of RSA key pair.
- The server certificate SHALL have the following attributes:
  - The SubjectDN parameter SHALL include CommonName attribute containing “127.0.0.1”.
  - The certificate SHALL contain SubjectAltName extension with at least following attributes:
    - extension is not critical,
    - dNSName: “localhost”, and
    - iPAddress: “127.0.0.1”.
  - The certificate SHALL contain KeyUsage extension with following attributes:

- extension is critical,
- digitalSignature key usage, and
- keyEncipherment key usage.

- The certificate SHALL contain ExtendedKeyUsage extension with at least following attributes:
  - extension is not critical, and
  - serverAuth key usage.

- Additionally, the certificate SHALL include any attributes that are specified in [PKIX] and [HTTPS].
- The SCS SHALL generate a self-signed root certificate from which the server certificate is issued. The root certificate SHALL contain attributes as specified in [PKIX].

2.4 CORS preflight check

A browser that supports CORS will do a preflight check to see if a CORS request is allowed to be sent to the SCS. This is done by sending an OPTIONS request to the web server indicating the Origin of the requesting HTML5 application, and the allowed request methods and headers. An example of such request is below.

```
OPTIONS /sign HTTP/1.1
Host: localhost:53952
Connection: keep-alive
Access-Control-Request-Method: POST
Origin: https://dvv.fineid.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
like
   Gecko) Chrome/36.0.1985.143 Safari/537.36
Access-Control-Request-Headers: accept, content-type
Accept: */*
Referer: https://dvv.fineid.fi/scstest/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2
```

As a response, the SCS should send an HTTP response back with specified CORS headers. An example of such a response is below.

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Server: SCS 1.2 (2021-12-01)
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers: Accept, Content-Type
Access-Control-Max-Age: 3600
Access-Control-Allow-Origin: *
Accept: application/json, */*
```

```
Date: Wed, 04 Jun 2017 07:38:01 GMT
```

2.5 Version check

2.5.1 Request

The HTML5 application checks the version of the SCS as well as discover the supported mechanisms, and availability of the SCS by sending an XMLHttpRequest to the SCS.

HTTP parameters:

- HTTP-Method: GET
- Request-URI: /version
- Other HTTP parameters are filled in by the browser as specified by HTTP/1.1 and CORS specifications.

Example HTTP request for version check.

```
GET /version HTTP/1.1
Host: localhost:53952
Connection: keep-alive
Accept: application/json, text/javascript, */*; q=0.01
Origin: https://dvv.fineid.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
like
   Gecko) Chrome/36.0.1985.143 Safari/537.36
Referer: https://dvv.fineid.fi/scstest/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2
```

2.5.2 Response

The SCS responses to the version check request by sending version check response.

HTTP protocol parameters:

- Status-Code: 200
- Content-Type: application/json
- Other HTTP parameters are filled in by the browser as specified by HTTP/1.1 and CORS specifications.

Version check response:

- **version** (string, mandatory): the version of the specification the SCS supports. For this specification the value is "1.2".
- **httpMethods** (string, mandatory): the allowed HTTP methods for signature creation request. The value for this specification is "GET, POST".
- **contentTypes** (string, mandatory): the supported data types supported by the SCS. The value for this specification is "data, digest".
- **signatureTypes** (string, mandatory): the supported signature types supported by the SCS. The value for this specification is
    - "signature,signature-plain,cms,cms-pades".
    - signature given data
        - If used key type is RSA, raw operation result is returned
        - If used key type is ECC, signature is ASN.1 DER encoded response if R and S integers as follows:
            - SEQUENCE { INTEGER = R, INTEGER = S }
    - signature-plain
        - If used key type is RSA, operation is compliant with ‘signature’
        - If key type is ECC, R and S components are returned as concatenated octet string: R || S
            - The length of R and S are padded to the same byte length as curve field length.
    - cms; if given content is
        - "data", cms structure becomes *attached*. Structure includes plain text data together with digital signature
        - "digest", cms structure becomes *detached*. Structure doesn’t include signed data.
    - cms-pades;
        - Designed to be used for PDF signatures [PADES].
        - Should be used in *detached* mode only.
- **selectorAvailable** (boolean, mandatory): determines whether the SCS supports the selector functionality. Possible values are "true" or "false".
- **hashAlgorithms** (string, mandatory): the supported hash algorithms supported by the SCS. The value for this specification is "SHA256, SHA384, SHA512".

- **signatureAlgorithms** (string, mandatory): the supported signature algorithms supported by the SCS. Possible values are "", "RSA", "RSASSA-PSS" and "ECDSA".
    - Example#1; if SCS contains only ECC keys, value is "ECDSA".
    - Example#2; if SCS contains only RSA keys but doesn't support RSASSA-PSS padding, value is "RSA".
    - Example#3; if SCS contains RSA and ECC keys and RSA key supports PSS-padding, value is "RSA,RSASSA-PSS,ECDSA".
    - Example#4; if there is no security token (like smart card) present, value is empty; "".
- **applicationOIDs** (string, optional): Display list of object identifiers introduced in "EF(DIR) Discretionary ASN.1 data objects". Currently used OIDs are listed in the table below.
- The version check can be used by the client application to discover the service, and validate that it is up and running.

Object Identifier	Card type
1.2.246.517.4.2.2.1	IAS ECC token for org. usage ("Toimikortti")
1.2.246.517.4.2.2.4	IAS ECC token for temp. usage ("Varakortti")
1.2.246.517.4.2.3.1	IAS ECC token for healthcare ("Ammattikortti")
1.2.246.517.4.2.3.2	IAS ECC token for healthcare ("Henkilostokortti")
1.2.246.517.4.2.3.3	IAS ECC token for healthcare ("Toimijakortti")
1.2.246.517.4.2.3.4	IAS ECC token for healthcare ("Varakortti")
1.2.246.517.4.2.3.5	IAS ECC token for healthcare students ("Opiskelijakortti")
1.2.246.517.4.2.4	**VRK FINEID ID.me org. card**
1.2.246.517.4.2.4.1	ID.me token for org. usage ("Toimikortti")
1.2.246.517.4.2.4.4	ID.me token for temp. usage ("Varakortti")
1.2.246.517.4.2.5	**VRK FINEID ID.me sote card**
1.2.246.517.4.2.5.1	ID.me token for healthcare ("Ammattikortti")
1.2.246.517.4.2.5.2	ID.me token for healthcare ("Henkilostokortti")
1.2.246.517.4.2.5.3	ID.me token for healthcare ("Toimijakortti")
1.2.246.517.4.2.5.4	ID.me token for healthcare ("Varakortti")
1.2.246.517.4.1	**VRK FINEID S4-1 v1.1 Citizen eID card (1.12.1999-31.3.2003)**
1.2.246.517.4.1.1	VRK FINEID S4-1-1 for KELA-FPA-SII (xx.xx.2000-31.12.2003)
1.2.246.517.4.1.2	VRK FINEID S4-1-2 v1.31 Citizen eID card (1.4.2003-->)
1.2.246.517.4.1.3	VRK FINEID S4-1-2 v1.31 Citizen eID card with sv-information (1.6.2004-->)
1.2.246.517.4.1.4	VRK FINEID S4-1 v3.0 Citizen eID card (1.1.2017-->)
1.2.246.517.4.1.5	VRK FINEID S4-1 v3.1 Citizen eID card (11.01.2021-->)
1.2.246.517.4.1.6	VRK FINEID S4-1 v3.2 Citizen eID card with ICAO-applet (03.08.2021-->)
1.2.246.517.4.1.7	DVV FINEID S4-1 v3.2 Citizen eID card with ICAO-applet (15.02.2023-->)

Example HTTP response:

```
HTTP/1.1 200 OK
Content-Length: xx
Content-Type: application/json
Server: SCS 1.2 (2021-12-01)
Access-Control-Allow-Origin: *
Accept: application/json, */*
Date: Wed, 04 Jun 2017 07:35:16 GMT

{
  "version": "1.2",
  "httpMethods": "GET, POST",
  "contentTypes": "data, digest",
  "signatureTypes": "signature,cms,cms-pades",
  "selectorAvailable": true,
  "hashAlgorithms": "SHA256, SHA384, SHA512",
  "signatureAlgorithms": "RSA,RSASSA-PSS,ECDSA",
  "applicationOIDs": "1.2.246.517.4.2.4.1,1.2.246.517.4.2.5.1"
}
```

2.6 Signature creation

2.6.1 Request (POST)

The HTML5 application makes a signature creation request by sending an XMLHttpRequest to the SCS.

HTTP parameters:

- HTTP-Method: POST
- Request-URI: /sign
- Content-Type: application/json
- Other HTTP parameters are filled in by the browser as specified by HTTP/1.1 and CORS specifications.

Signature creation request parameters:

- **version** (string, optional) contains the version of the SCS specification version that the HTML5 application expects the SCS to support (which for this version is "1.2").
- **selector** (object, optional) contains the certificate selector parameters
    - **issuers** (array, optional) a list of acceptable issuers of the end user certificate in string format (e.g., "CN=Trusted CA, OU=Unit, O=Organization, C=FI") or in ASN1/DER encoded form as it is present in the end user certificate (i.e., the issuer field in the end user certificate [PKIX]) in which case the format is “base64:<ASN1/DER encoded issuer field as it is present in the end user certificate in

base64 encoded format>". Values are case sensitive. If specified, the end user certificate MUST have one the listed issuers as the issuer.

It is recommended to pre-filter the certificate list using the "akis" array field instead of the "issuers" array field.

- **akis** (array, optional) a list of authority key identifiers of acceptable issuers in base64 format (key identifier is the content of the "keyIdentifier" field in the AuthorityKeyIdentifier extension in the end user certificate as specified in chapter 4.2.1.1 [PKIX], i.e., either the SHA-1 hash of the authority certificate's public key or 60 least significant bits of this hash preceded by '0100' bits as specified in chapter 4.2.1.2 in [PKIX]). Values are case sensitive. If specified, the end user certificate MUST have one of the listed keyIdentifiers present in the AuthorityKeyIdentifier extension.

- **keyusages** (array, optional) a list of required keyusages as they are listed in KeyUsage extension ("digitalSignature", "nonRepudiation", "dataEncipherment", "decipherOnly", "encipherOnly", "keyAgreement", "keyEncipherment", "keyCertSign", "crlSign"). Values are case insensitive. If specified, the end user certificate MUST have all the listed key usages present in the KeyUsage extension.

- **keyalgorithms** (array, optional) a list of acceptable key algorithm of the end user certificate ("rsa", "ec"). Values are case insensitive. If specified, the end user certificate SHOULD have one of the listed key algorithms as type of the subject public key in the end user certificate.

- **strictKeyPolicy** (Boolean, optional). This selector is used with together with **keyalgorithms**. If the value is true, selected end user certificate MUST have one of the listed key algorithms as type of the subject public key in the end user certificate.

- **content** (string, mandatory) contains the data to be signed that is base64 encoded.

- **contentType** (string, optional) specifies the type of the data field. Default is "data". Options for type are "data" meaning that the data field contains the data that should be signed, and "digest" meaning that the data field contains the digest of the data that should be signed. Supported values are "data" and "digest".

  - For authentication purposes, contentType must be "data", "digest" is not allowed.

  - For signing purposes, contentType can be either "data" or "digest".

- **hashAlgorithm** (string, optional) specifies the requested signature algorithm. Default is "SHA256" indicating "SHA256withRSA" signature with RSA keys, and "SHA256withECDSA" signature with EC keys. Supported values are "SHA256", "SHA384", and "SHA512".

NOTE: If the contentType parameter value is "digest", the data parameter must contain a digest calculated using the digest algorithm indicated in the algorithm parameter.

- **signatureAlgorithm** (string, optional) specifies the requested signature algorithm. Supported values are “RSA”, “RSASSA-PSS” and “ECDSA”. Default algorithm is “RSA” or “ECDSA” depending on used key type.
  - “RSA” stands for RSASSA-PKCS1-v1_5 signatures [PKCS1]
  - “RSASSA-PSS” stands for PKCS#1 v2.1 padding scheme that is suggested to be used with new implementations
  - “ECDSA” for EC keys [ECDSA].

- **signatureType** (string, optional) specifies the requested signature format. Supported values are "signature", “cms” and “cms-pades”.
  - Default is "signature" indicating plain raw signature
  - Value “cms” indicates that signature is based in CMS signature format [CMS].
    - The signed data is included (attached) to the CMS signature when contentType is “data” or
    - signed data is excluded (detached) from the CMS signature when contentType is “digest”. See CMS content profile in section 4.
  - Value “cms-pades” indicates that signature is based on CMS signature format with exact attributes defined in ETSI standard for Advanced Electronic Signatures [PADES].
    - This signature type should be used with “digest” (detached) contentType only.

Example HTTP POST request of signature creation request:

```
POST /sign HTTP/1.1
Host: localhost:53952
Connection: keep-alive
Content-Length: 155
Accept: application/json, text/javascript, */*; q=0.01
Origin: https://dvv.fineid.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
like
   Gecko) Chrome/36.0.1985.143 Safari/537.36
Content-Type: application/json
Referer: https://dvv.fineid.fi/scstest/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2

{
  "version": "1.2",
  "selector": {
    "issuers: [
```

```
      "CN=VRK CA for Qualified Certificates, ..., C=FI",
      "CN=VRK CA for Qualified Certificates - G2, ..., C=FI"
    ],
    "keyusages": [
      "nonrepudiation"
    ]
  },
  "content": "VGhpcyBpcyB0aGUgZGF0YSB0byBiZSBzaWduZWQuLi4=",
  "contentType": "data",
  "hashAlgorithm": "SHA256",
  "signatureType": "signature"
}
```

2.6.2 Request (GET)

The HTML5 application may send the signature creation request also using GET method. The HTTP protocol and request parameters are the same as with POST method with following exceptions:

HTTP parameters:

- Request-Method: GET

Signature creation request parameters:

- **selector** functionality is not supported with GET method.

Example HTTP GET request of signature creation request:

```
GET
/sign?content=VGhpc...QuLi4=&hashAlgorithm=SHA256&contentType=digest
HTTP/1.1
Host: localhost:53952
Connection: keep-alive
Accept: application/json, text/javascript, */*; q=0.01
Origin: https://dvv.fineid.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
like
   Gecko) Chrome/36.0.1985.143 Safari/537.36
Referer: https://dvv.fineid.fi/scstest/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2
```

2.6.3 Response

The SCS responses to the signature creation request by sending the signature creation response.

HTTP protocol parameters:

- Status-Code: 200

- Content-Type: application/json
- Other HTTP parameters are filled in as specified by HTTP/1.1 and CORS specifications.

Signature creation response:

- **version** (string, mandatory) contains the version of the SCS specification version that the SCS supports (which for this version is "1.2").
- **status** (string, mandatory) indicates whether the signature creation operation was successful.
    - "ok" indicates that the operation was successful,
    - "failed" indicates that the operation failed.
- **reasonCode** (integer, mandatory) gives the reason code of the response status. Possible values are listed in section 0.
- **reasonText** (string, mandatory) gives textual description of the reason code.
- **signature** (string, optional) contains the digital signature that is base64 encoded. Present if signature creation operation was successful.
- **signatureType** (string, optional) contains the type of the signature. Supported values are "signature", “cms” and “cms-pades”. Present if signature creation operation was successful.
- **signatureAlgorithm** (string, optional) contains the format of the digital signature. Supported values are:
    - "SHA256withRSA"
    - "SHA384withRSA"
    - "SHA512withRSA"
    - "SHA256withRSASSA-PSS"
    - "SHA384withRSASSA-PSS"
    - "SHA512withRSASSA-PSS"
    - "SHA256withECDSA"
    - "SHA384withECDSA"
    - "SHA512withECDSA"
    - Not present or empty value if operation failed
- **chain** (array, optional) contains the certificate chain of the end user certificate. The end user certificate is at index 0. All certificates are base64 encoded.

Present if signature creation operation was successful. The chain may be incomplete and contain only the end user certificate.

Example HTTP response of signature creation response:

```
HTTP/1.1 200 OK
Content-Length: 5858
Content-Type: application/json
Server: SCS 1.2 (2021-21-01)
Access-Control-Allow-Origin: *
Accept: application/json, */*
Date: Wed, 04 Jun 2017 07:38:16 GMT

{
  "version": "1.2",
  "signatureAlgorithm": "SHA256withRSA",
  "signatureType": "signature",
  "signature":
    "eHF1oXKD62+KTSUb3GPNMhYumwjtL...EVvvo++k+fzDK41nvK5lAZSkA==",
   "chain":[
      "MIIGADCCBOigAwIBAgIEBfetjDA...iLniyk0+r5aasoF7TY9n4tpMWmg==",
      "MIIFcDCCBFigAwIBAgIDAdbDMA0...fBVhROQzKoDAk1DPtg1gOADYfLg==",
      "MIIEHjCCAwagAwIBAgIDAdTAMA0...lEtXfFCfv9DA8MeVIUfa2pTH6Tk="
   ],
  "status": "ok",
  "reasonCode": 200,
  "reasonText":"Signature generated"
}
```

2.7 Transactions

Transactions are designed to fulfill following requirements:

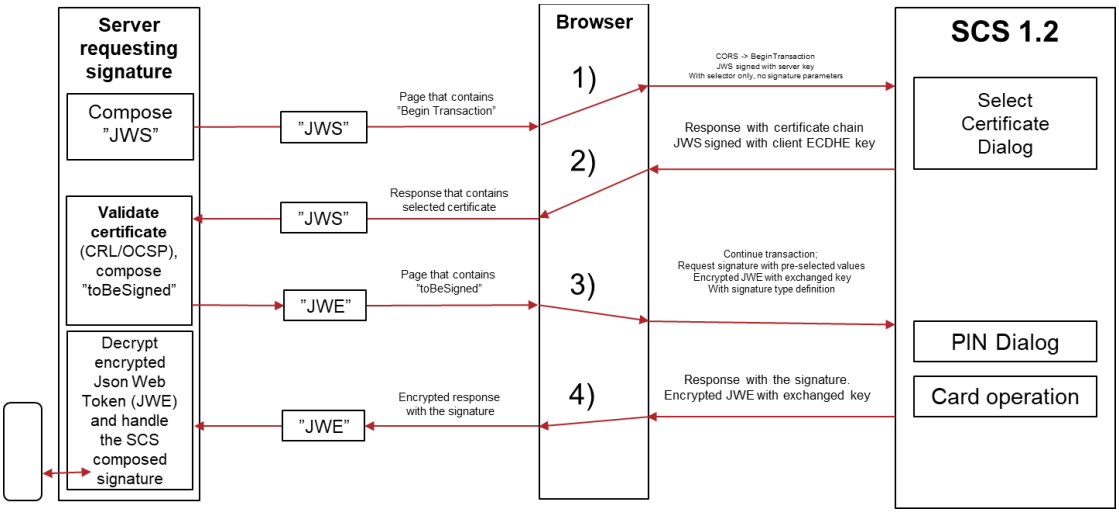
1) Allow SCS to see who is requesting signature

2) Protect communication between SCS and server

3) Allow server to query certificate from SCS before server composes actual signature request.

Message protection is based on signed and encrypted Json Web Tokens [JWT]. First two “begin transaction” messages are signed (JWS) messages, followed by two “execute transaction” messages that are encrypted according JWE.

Message payload follows this specification.



2.7.1 Begin transaction

“Begin transaction”-phase allows the SCS user to confirm from where the signature request was generated. Also, during this phase server and client executes key exchange routine, that is used to encrypt messages in the “execute transaction”-phase in the way that only client and server can decrypt transmitted data.

Begin transaction messages are composed as follows:

1) server generates ephemeral EC keypair (ECDHE)

2) server composes payload JSON message that contains following claims; ***version***, ***selector***, ***contentType***, ***hashAlgorithm***, ***signatureType***, ***content***, ***serverKey, serverCert*** claims are used in this phase.

- ***version***, ***selector***, ***contentType***, ***hashAlgorithm, signatureAlgorithm*** and ***signatureType*** are encoded as defined in the chapter 2.6.1.
- ***serverKey*** string object (generated ephemeral EC keypair) should be encoded as follows;
    - Value is base64 encoded public key according ANSI X9.62, chapter 6.4, Syntax for Public Keys [X9.62] and SEC 1, chapter C.3, Syntax for Elliptic Curve Public Keys [SEC1].
        - Suggested ***ECDHE*** curve type is ***secp256r1***.
- ***content*** string object is UTF8 encoded string that will be displayed during signature request.
- ***serverCert*** string object is base64 encoded binary array that contains DER encoded server certificate. This certificate is used to validate JWS.

3) Server signs payload with its private key.

Example “Begin transaction” message:

```
POST /Sign HTTP/1.1
Host: localhost:53951
Connection: keep-alive
Content-Length: 2651
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45
Safari/537.36
Content-type: application/jwt
Accept: */*
Origin: http://127.0.0.1:53951
Referer: http://127.0.0.1:53951/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fi;q=0.8

eyAiYWxnIjogIlJTMjU2IiwgImtpZCI6ICJGMEE0RDlCQzMwMTdENUM3RDgwOENDNDlBODk5RUI4QzU4NDM0NzVGIiwgInR5cCI6ICJKV1QiIH0.eyA
idmVyc2lvbiI6ICIxLjIiLCAic2VsZWN0b3IiOiB7ICJrZXlhbGdvcml0aG1zIjogWyAicnNhIiwiZWMiIF0sICJrZXl1c2FnZXMiOiBbICJub25yZX
B1ZGlhdGlvbiIgXSB9LCAic2VydmVyS2V5IjogIk1Ga3dFd1lIS29aSXpqMENBUVlJS29aSXpqMERBUWNEUWdBRVJOVVB5ZDh5djhTSjk4VFBZcFJxd
kRyT2VxMEUzMldhL2pRc1hnUXpRaXZNdi9nWStnS25GUVZLNHd0T1VZWVRNTHl6YzlaYU1YaGhJcHkzTTE5ZDZRPT0iLCAic2VydmVyQ2VydCI6ICJN
SUlFQ0RDQ0F2Q2dBd0lCQWdJSW1wY3AwcXhVZS9vd0RRWUpLb1pJaHZjTkFRUxCUUF3VXpFTE1Ba0dBMVVFQmhNQ1Jra3hHekFaQmdOVkJBb1RFa1o
xYW1sMGMzVWdSbWx1YkdGdVpDQlBlVEVuTUNVR0ExVUVBeE1lUm5WcWFYUnpkU0JFYVdkcFUybG5iaUJNYjJOaGJDQlNiMjkwTFVOQk1CNFhEVEl4TU
RreU56SXpOVGsxT1ZvWERUTXhNRGt5TlRJek5UazFPVm93VnpFTE1Ba0dBMVVFQmhNQ1Jra3hHekFaQmdOVkJBb1RFaloxYW1sMGMzVWdSbWx1YkdGd
VpDQlBlVEVyTUNrR0ExVUVBeE1pUm5WcWFYUnpkU0JFYVdkcFUybG5iaUJNYjJOaGJDQkRaWEowYVdacFkyRjBaVENDQVNJd0RRWUpLb1pJaHZjTkFR
RUJCUUFEZ2dFUEFEQ0NBUW9DZ2dFQkFMUEpmNnVEdXljZGIxR1lYWllRZ1BZamVySGVRdXZDem5UMm1IdFFPVDllaWxISG5RT3haOUY2Vzl3L24xRFV
IbW1RNjZoeEVlTExHYXBzUXVUVUMxQXN0RHBmK0U4dDE0MzJDbHIwZjZPd2tJaklqZTJBYXYrR2o2VXRGZnVieU1nQlVOTi9oV2FZTW11WitpNmFuY0
N0cUtBa3FQempDRDdhYXJVOTNESnpFYzBMTHdEWktHY21MN2IycVd2WTU0OVB5S0EwT0JicHRoeU4ySzJjMURQT0tWNlUzcThURjFQMWVyY3M3MSszV
jJLdytGdUQyRDNXdzI3TVl2ckx1MnFlSVJiK0dFZFB0QVQ4aGNaSGVJdmd6Z1cvU1ZxVldVOUFvWDVIS2gzU3AxdXRiZDJxa2Q2WjZPRUZMV01uWmlK
UzhiQTVIMERpelpvandaUkV6b2NDQXdFQUFhT0IyekNCMkRBZEJnTlZIU1VFRmpBVUJnZ3JCZ0VGQlFjREFRWUlLd1lCQlFVSEF3SXdEZ1lEVlIwUEF
RSC9CQVFEQWdXZ01DVUdBMVVkRVFRZU1CeUNDV3h2WTJGc2FHOXpkSUlKTVRJM0xqQXVNQzR4aHdSL0FBQUJNRUFHQ0NzR0FRVUZCd0VCQkRRd01qQX
dCZ2dyQmdFRkJRY3dBb1lrYUhSMGNEb3ZMMnh2WTJGc2FHOXpkRG8xTXprMU1TOXpjMnhwYzNOMVpYSXVZM0owTUI4R0ExVWRJd1FZTUJhQUZDdUkzS
Ew5WDlpVzdtZXgvNDIrR1hKOEFkbTRNQjBHQTFVZERnUVdCQlR3cE5tOE1CZlZ4OWdJekVtb21ldU1XRU5IWHpBTkJna3Foa2lHOXcwQkFRc0ZBQU9D
QVFFQU1IVDRpeXowcUd4Ymw0Y09DME4xR0JwaEhqQVBXZDcxMzVPTGdnbzBNS3EzNElhOVZpeGd5OHZlNDFCTU53SktMZ1lwc3QzaVdvMVRFV1ltUnN
iYVA2SHA2TWNpanhQdGJXMjZSOHdxbFhnUnpxNCtvQVdtczhsRHpIbmp0L1k2ako3MWJhcld5aHVMRVNsOTFuRmh1Y0M4aVNPazF4S0JleWxwZmlGMW
lPcXplQ2luc1JNOEs3ZzNiQkVRWUloWVZNeFB2UVgybm9EdC82YU5ucjJHTGNycER3MDFoc3Q5amRlVGhGRzZBaHVBMWhpcXFNRkkyajJxN1Z6dUM1M
0FFSTdPVWRuU0RrTmFOeEhBK1VlWFBaTDlGZmRtQUN0aGt0Q1JWb0Y0bmRIY2lDOS9nUTRRelJpdnpta1pVcDc0bEFiSVlSY2g5aUg4ZzNoeSswQ2ta
dz09IiB9.SDckvDq-hGOeU4NvEUtDkK45Y2K-
z1PklbBh47pCMWJJan_e6Z8Tpleu_z9gPAjmuTZ32w3EZbAlOx4PLNKOG9332u24LvZ56ExtLOrvsEKh13yDAMxMCz4XhnuoRIFULb3oM1ytbrsLvoZ
HD042ZL_RHzFjhsfi2oIJ42j-Pf6pI9lGFXBQwLB5ypSMEVL-
ob45M25DFYoSunlcCTVgqb5nwncltHxEBDKt4R7C6abfxY96GqJWekInnIZ4NWEONQLZibRgB6HKTNW9YNaGbOF5fCH37oHUyYzKlo97eNuXSR-
5lM2WqZVvvNPfV1g42ZDbrPxnmUUHUcrxbZNF4w
```

JWS Header:

```
{
  "alg": "RS256",
  "kid": "BC9CDE58A009D52BFA065FC2114E19188CDED8BD",
  "typ": "JWT"
}
```


JWS Payload:


```
{
  "version": "1.2",
  "selector": {
    "keyalgorithms": [ "rsa","ec" ],
    "keyusages": [ "nonrepudiation" ]
  },
  "serverKey": "MFkwEwYHKoZIzj...",
  "serverCert": "MIIECDCCAvCgAwI..."
}
```


Request is validated by SCS and it MUST ask user confirmation to continue.

- After confirmation, response is sent back to caller with SCS side ECDHE key and certificate chain.
- ***transaction*** string object contains ephemeral ECDH public key of SCS side.
    - Value is base64 encoded public key according ANSI X9.62, chapter 6.4, Syntax for Public Keys [X9.62] and SEC 1, chapter C.3, Syntax for Elliptic Curve Public Keys [SEC1].
- ***chain*** (array, mandatory) contains the certificate chain of the end user certificate. The end user certificate is at index 0. All certificates are base64 encoded.

Example response message:


```
HTTP/1.1 200 POST
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://127.0.0.1:53951
Content-Type: application/jwt
Content-Encoding: gzip
Host: localhost
Content-Length: 6601

eyAiYWxnIjogIkVDMjU2IiwgImtpZCI6ICI2MTczREI2RUU2OERBNjRGMjY0MzVGMjU1RTMzQzJFMUI1OTU2RUNBIiwgInR5cCI6ICJKV1QiIH0.eyA
idmVyc2lvbiI6ICIxLjIiLCAidHJhbnNhY3Rpb24iOiAiTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJemowREFRY0RRZ0FFc04weFMrZjNrSFBxay8rRk
NJRlFPSnNSRjYzQUN1cy81R1UrbENjZ2Z4UlJ3SU5xc2JTWncxTGI3U1pIZFJxWGcyMWFYd3NjeW5ZajZOd2IwMm1pOHc9PSIsICJjaGFpbiI6IFsgI
klJSUhLekNDQlJPZ0F3SUJBZ0lFQmd6R2xEQU5CZ2txaGtpRzl3MEJBUTBGQURCNUlRc3dDUVlEVlFRR0V3SkdTVEVqTUNFR0ExVUVDZ3dhVm1GbGMz
UnZjbVZyYVhOMFpYSnBhMlZ6YTNWeklGUkZVMVF4R0RBV0JnTlZCQXNNRDFSbGMzUnBkbUZ5YldWdWRHVmxkREVyTUNrR0ExVUVBd3dpVmxKTElGUkZ
VMVFnUTBFZ1ptOXlJRlJsYzNRZ1VIVnljRzl6WlhNZ0xTQkhOREFlRncweE9UQTFNakF4TVRJNE1qTmFGdzB5TkRBeU1qVXlNVFU1TlRsYU1IMHhDek
FKQmdOVkJBWVRBa1pKTVE0d0RBWURWUVFLREFWVWRXeHNhVEVTTUJBR0ExVUVCUk1KT1RrNU1EUTVOekV3TVE0d0RBWURWUVFxREFWRVJVNUpVekVVT
UJJR0ExVUVCQXdMU2xWVlZFbE1RVWxPUlU0eEpEQWlCZ05WQkFNTUcwcFZWVlJKVEVGSlRrVk9JRVJGVGtsVElEazVPVEEwT1RjeE1EQ0NBYUl3RFFZ
SktvWklodmNOQVFFQkJRQURnZ0dQQURDQ0FZb0NnZ0dCQU1oWU0wVTNzelV2aURvbi9sTHFEdWdITG9zT2NjczhoaGJGOHdpS1JSei9MSHBWWTRZczN
2SXVtMmNjNGZ6MWhMZzMwaGtaRDUwWHhpY3FIOTJXQ1ZrbXdWNm5YZWk2bkp6aE5JdTdBeVgrZm5aeGdIQ1JwZHlrY3h1VWdjeEJjaTRsWG5EcHBrZF
ViSnZidkY1UmFYa2xKeHBua2dDVW5NSzJhU1FYQ21DaDAvblNmRERxM29jQkVCaHBKOHpRK3JTQ0dkcUdidXBlZ0d3RU4zY3pIWGJMTnJPOXFmMVVsM
```

```
WRnQXNqNENrMFBLOU9pY2o2bnhnR0xiZmo5Zjh5RG5FVmZJYUdhUGlhVEI5NSsxRkFsZVRzTVQ4WVNZQWMyN0N3RmNIaGRkMDFwTHFiOGpzRGg5TUFz
MWg5MnF0bUxHQ2UyWHdUazRxbjE5NGFjREN3bmdlZU95d2tSYkF4bEdndjdtWG1sMTNpL0VIamp6M1NvWUlUb1BmZVY3SVluRkl5dTV6dHYwQ3g5Z1N
yaUNwb3dNL0ErUjVBSTBDVlVQeEFSTTNGUGN2bjA5cHcydndFY1VrTjZ1amRCUG80WjV5K0hRYmtWeHcxckNTeXdjV0dxVXFQMFFoVFZNZDRPaXJjWk
VodlFkeTNPTmZXRUd5Y3FUMkg4amY3WnB3NVYxUUlEQVFBQm80SUNOVENDQWpFd0h3WURWUjBqQkJnd0ZvQVVQWnFqdGZnVkVlOFJ5dXZIWEUyVGdMT
EhQOEV3SFFZRFZSME9CQllFRkxGa3d4TGMzZXlFZThiM1ExNnBzWUxicFQ1ZU1BNEdBMVVkRHdFQi93UUVBd0lHUURRQnpnWURWUjBnQklIR01JSERN
SUhBQmdvcWdYYUVCV01LZ1VzQk1JR3hNQ2NHQ0NzR0FRVUZCd0lCRmh0b2RIUndPaTh2ZDNkM0xtWnBibVZwWkM1bWFTOWpjSE01T1M4d2dZVUdDQ3N
HQVFVRkJ3SUNNSGthZDFaaGNtMWxibTVsY0c5c2FYUnBhV3RyWVNCdmJpQnpZV0YwWVhacGJHeGhJQzBnUTJWeWRHbG1hV3RoZENd2IyeHBZM2tnWm
1sdWJuTWdMU0JEWlhKMGFXWnBZMkYwWlNCd2IyeHBZM2tnYVhNZ1lYWmhhV3hoWW14bElHaDBkSEE2THk5M2QzY3VabWx1Wldsa0xtWnBMMk53Y3prN
U1DVUdBMVVkRVFRZU1CeUJHbVJsYm1sekxtcDFkWFJwYkdGcGJtVnVRSFIxYkd4cExtWnBNQThHQTFVZEV3RUIvd1FGTUFNQkFRQXdOd1lEVlIwZkJE
QXdMakFzb0NxZ0tJWW1hSFIwY0RvdkwzQnliM2g1TG1acGJtVnBaQzVtYVM5amNtd3ZkbkpyZEhBMFl5NWpjbXd3YmdZSUt3WUJCUVVIQVFFRVlqQmd
NREFHQ0NzR0FRVUZCekFDaGlSb2RIUndPaTh2Y0hKdmVIa3VabWx1Wldsa0xtWnBMMk5oTDNaeWEzUndOQzVqY25Rd0xBWUlLd1lCQlFVSE1BR0dJR2
gwZEhBNkx5OXZZM053LmZEdWemRDNW1hVzVsYVdRdVpta3ZkbkpyZEhBME1DMEdDQ3NHQVFVRkJ3RURCQ0V3SHpBSUJnWUVBSTVHQVFFd0V3WUdCQUNPU
mdFR01Ba0dCd1FBamtZQkJnRXdEUVlKS29aSWh2Y05BUUVOQlFBRGdnSUJBRDcrV0FhdjNXZkRZMk4zV0lpWHRncFN3bUUvYUhZU0hCc2tVKyttSzVV
azNCMWRFNFdHYStrdTFocTNXdktOUTZQdHRHcTdXY0dXcGRVNFNWOGhwWUNRS2NUSmV2WVlvYzlXa040WkQ2RW5jRkovelMwMC9jT1pSUWx3SXhkbzV
FbXlZaWNXeUFOZktqMTlQTGxWYjZFTHFLUkcxdmxFSWlwQ0pyUmdGUGNKaitDdldzeFoxZDhOMDJuS2FqbUdwRzQ0M3g3SjdvTHhYTldPZ0JTNlJzVT
I2MkNkbjlPS2t5S3BnNWlYaXA0UHlneVBJSjZjY28wcFl3RCsyenFwYm11d0RHMlFMR0dpK1k1VDUxQzJ2THZnbEZsS0txKzJLQm05ZlJYL2VoR044U
2hqN2VvdjFNZVBBRTd2bnRNQ2E0RlkyaDloN1UyNitaNmNSUWl1WUlsTSt4cWlueHowL0xpa2RScHU0R0RyNjJiUUd4UDVCNDJPVXlTbHl6TWVRTHNl
b0w3TkhPdHI0S1JWYUZENHFwNjVQSXlxNTJTUW90WmxUSFJxU1NoRlBvRW40YmZXUDlvaDROR1JrMHVLY0ZMckl0RXBpeHVRcis4ZVlEcG1rMWRNSTB
yczQyUWVsVDBnQngyQTkvYVVHNnZWcXFaenBPTnpaTE9KOUpFSGxtQVN1K1IvUFNQdHBMMUtzYXl3WnVwcUpXeEdZTnZ0aXZYclN6Tks1Ui9pa3laTm
taMGlEUnRxYkZra0Q4anBPbUk1YTJ4ZGN6STBmNzA3cnhKTGlYOFRENUNjLzdqdjk2K3pxU2pzbWoyenNRa2l0ZmxaL09PcmNTZ3plYzU4OUczVyt4a
lZCdmtTb0h4MW1MdGhDM2phN0VpRVY3L3RBLzR2bWRvYkM5S0UiLCJNSUlIZXpDQ0JXT2dBd0lCQWdJREExdGpNQTBHQ1NxR1NJYjNEUUVCRFFVQU1J
R1lNUXN3Q1FZRFZRUUdFd0pHU1RFak1DRUdBMVVFQ2d3YVZtRmxjM1J2Y21WcmFYTjBaWEpwYTJWemEzVnpJRlJGVTFRReEtUQW5CZ05WQkFzTUlFTmx
jblJwWm1sallYUnBiMjRnUVhWMGFHOXlhWFI1SUZObGNuWnBZMlZ6TVJrd0Z3WURWUVFMREJCV1lYSnRaVzV1WlhCaGJIWmxiSFYwTVI0d0hBWURWUV
FEREJWV1Vrc2dWRVZUVkNCU2IyOTBJRU5CSUMwZ1J6SXdIaGNOTVRnd01qQTVNVEF3TmpMFdoY05Nemd4TURFd01UQXdOakkwV2pCNU1Rc3dDUVlEV
lFRR0V3SkdTVEVqTUNFR0ExVUVDZ3dhVm1GbGMzUnZjbVZyYVhOMFpYSnBhMlZ6YTNWeklGUkZVMVF4R0RBV0JnTlZCQXNNRDFSbGMzUnBkbUZ5YldW
dWRHVmxkREVyTUNrR0ExVUVBd3dpVmxKTElGUkZVMVFnUTBFZ1ptOXlJRlJsYzNRZ1VIVnljRzl6WlhNZ0xTQkhORENDQWlJd0RRWUpLb1pJaHZjTkF
RRUJCUUFFZ2dJUEFEQ0NBZ29DZ2dJQkFNYUN0RXJBaStZYW5wM2pZeHBFdUtCd0JrZWYwek9LSGJ5czhtM3N2SWQ3aXU2eHBWb3ljUTBrR2w1TUxYSW
pSdUNOTWhRbkEvcFdZbjczLzRnd3pRMFpXdkwzbjNIaWptcUd1aDFraHNXSElDZDl5Q2pVR0FNT2MweXdsc25QNzhvcEhwakNDT3JqYkkzZGtVQkRub
VFvY1NGUG95SG4wQ1IyRitjaDNlbUpWZGRuc29Way9FSkl5ejZ0NmFtcUkzVVVrMGJnY0VqakU2SEM1aDVVOHFHVEFsVzBLQzFRanRZQnRtV1c4MWVh
czZKMU4xaHp6MlJWWHZmc09ZbjF3eUFZQi9DYmE5Zkk2MEE2TXZGcWgvbUJXc1hXblgzL3JwdzgzWlMwTmFmWlcvVGV1Z0hhdStjaVRlcGxPRGE2eHp
zVUEreFhWeW4zWkwvcUN2WnBnVldjek42OFpOMzY5SGRDMSs3ZWE3UzIzYXkxb2t5MW9nbjlVaEVCNGlyRnpFcjJjS09ZWEhITEFEbUhORllnRUkxa0
VoUmVBbFRNNGZKbm5kZ0dZRGMrRCs5aFJiK1R2dHMyWGRxdVVndnJtNVczbnNxOTRpbTRNYWdJd2pQZHZpc1hMSDM2MTk2Rk01WllaWExxaTdQQkd5a
kczQ2dMQmpZWVZQaDZHKzJkSHBrd0x6bFJ4UTRnczBlaEk0L1B2TzNTb05vMzI5NE9aQUsyS3RxUzM2ZW1aakFEK1RJaWJtbFNhbXhYOXhHMkYyZTJX
Z1lJNXk0NmcvN0NXa1dIR3pCMWJablM1aEhoM2tvMDZTalNMM1VQRS9tL2pYOEZvc2FXbE1ocDJIMGx2RlRmckxYeDhKbWxjWCtramlDdlF2MDcwb3F
yZmtDZUNxMVh6cno3ZEdvWEFnTUJBQUdqZ2dIcU1JSUI1akFmQmdOVkhTTUVHREFXZ0JTRlBhRGgvTEVKSitUYXZSK0dpMEFHY2tJS2dUQWRCZ05WSF
E0RUZnUVVVQWnFqdGZnVkVlOFJ5dXZIWEUyVGdMTEhQOEV3RGdZRFZSMFBBUUgvQkFRREFnRUdNSUhPQmdOVkhTQUVnY1l3Z2NNd2djQUdDaXFCZG9SR
ll3cUJTTUV3Z2JFd2dVUdDQ3NHQVFVRkJ3SUNNSGthZDFaaGNtMWxibTVsY0c5c2FYUnBhV3RyWVNCdmJpQnpZV0YwWVhacGJHeGhJQzBnUTJWeWRH
bG1hV3RoZENd2IyeHBZM2tnWm1sdWJuTWdMU0JEWlhKMGFXWnBZMkYwWlNCd2IyeHBZM2tnYVhNZ1lYWmhhV3hoWW14bElHaDBkSEE2THk5M2QzY3V
abWx1Wldsa0xtWnBMMk53Y3prNU1DY0dDQ3NHQVFVRkJ3SUJGaHRvZEhSd09pOHZkM2QzTG1acGJtVnBaQzVtYVM5amNITTVPUzh3RWdZRFZSMFRBUU
gvQkFnd0JnRUIvd0lCQURBNUJnTlZIUjhFTWpBd01DNmdMMS0FxaGlob2RIUndPaTh2Y0hKdmVIa3VabWx1Wldsa0xtWnBMMk5oTDNaeWEzUndNbTVuY
W1FdVkzSnNNNSFFHQ0NzR0FRVUZCd0VCQkdnd1pqQXZCZ2dyQmdFRkJRY3dBWVlqYUhSMGNEb3ZMM0J5YjNoNWNsTWFyYTjBMbVpwWm1WcFpDNW1hUzky
MFpYTjBNbU13TXdZSUt3WUJCUVVITUFLR0oyaDBkSEE2THk5d2NtOTRlUzVtYVc1bGFXUXVabWt2WTJFdlZuSnJkR1l6ZERKakxXTnlkREFOQmdrcWh
raUc5dzBCQVEwRkFBT0NBZ0VBRkhpODVURnF0V1I4bWJDREdhcExMUjRmRjE0dWJ5eWxBQ1JESHBKYXAyTGZTOHAxTEIyRmtoRGFzdnFITERpWUNaaE
huWkFyd0hpaEpCcmlVbGFPY2ZjaHd5WkxYSnNFdERScWRNdTBjTU9sYkt1QU9uZkFsdnU2SnFybXdtemhOTnNJNTJhME1obFA3TDVRd3oyU3VpWkJLR
U00NnB5VXIyU3JaTVAzZG00WWNXQi91UUpoMGtNbS95TWQwbEp0Mm45R0owUDlKTUtlelNlYmNVb0MycFFyeUhZaElTcTVaSDI4ZFY5SkNyN1EvbnZN
eForY3cyaFJzY1RtdS9mYlk3bXIxNm5mQXNiaHM4TjhnQ2lEWjArUEpXZE9ydHowZDJvaGt6VnhZRlhFTTdRSjM2aEhKNS9ja005TGFsdi9iTzhUUmF
HM2hlRFJPT1lSTjFOdnFFRXYrazFmYmdEZVdJOHdMcDl4ZmhOZVB5cTVjRmpWczFtRXV4VkxSSFEwOGxDQ21IcG14bUsxK0g2REJSTVRtMGI1VU9tMz
FpRnFMOXQ4VFJGeGhBelBuY1JQNXZwcjBYWDdZTEt1YUJ5VmNIR1FGemZMUHZLdEx6T3JZRy81Z01Kb1FaU3hyU0JmVVZwbis2Uml3aytKMjF1bUdsN
2FzdHkxNFc5UzBRSVFJTjNEdHlhbTdXdGtGYVEwOW1uZFZ4bGNKTGVpZjRITWsrWWp5Unh4Ym44L3FtNjkva0ZyVlNZVkhGRE5SVHpOM2d1a2NSWCtp
ckFoRVZzRnJWUW1CQTRieHptZWVIOEc0Y0p3Y0dzS3ovZG93eEozQnVWMjFNcExxIRzZZUjZtdElYVWJaeHpzamRIb3RvaHRaeG1KNm4yeTVuUXRHVzd
FNldKanlSTnhRYjA9IiwiTUlJR0VqQ0NBL3FnQXdJQkFnSURBMXRnTUEwR0NTcUdTSWIzRFFFQkRRVUFNSUdZTVFzd0NRWURWUVFHRXdKR1NURWpNQ0
VHQTFVRUNnd2FWbUZsYzNSdmNtVnJhWE4wWlhKcGEyVnphM1Z6SUZSRlUxUXhLVEFuQmdOVkJBc01JRU5sY25ScFptbGpZWFJwYjI0Z1FYVjBhRzl5Y
VhSNUlGTmxjblpwWTJWek1Sa3dGd1lEVlFRTERCQldZWEp0Wlc1dVpYQmhiSFpsYkhWME1SNHdIQVlEVlFRRERCVldVa3NnVkVWVFZDQlNiMjkwSUVO
QklDMGdSekl3SGhjTk1UY3hNREV4TVRBd016TXpneE1ERXhNVEF3TXpNeVdqQ0JtREVMTUFrR0ExVUVCaE1DUmtreEl6QWhCZ05WQkFvTUd
sWmhaWE4wYjNKbGEybHpkR1Z5YVd0bGMydDFjeUJVVUlZOVU1Ta3dKd1lEVlFRTERDQkRaWEowYVdacFkyRjBhVzl1SUVGMWRHaHZjbWwwZVNCVFpYSj
JhV05sY3pFWk1CY0dBMVVFQ3d3UVZtRnliV1Z1Ym1Wd1lXeDJaV3gxZERFZU1Cd0dBMVVFQXd3VlZsSkxJRlJlJGVTFRZ1VtOXZkQ0JEUVNBdElFY3lNS
UlDSWpBTkJna3Foa2lHOXcwQkFRRUZBQU9DQWc4QU1JSUNDZ0tDQWdFQXBkMXlHZ2g5NmNtRDlLVnlPclBEMmVSaWM4NjBLRmp0S1ZHQ3hZNXA2MTNO
dy9ocHg0NnluSFNBMHpaQWtCdzFyUkFaMTgvTUNVZnE4K1lrWVExTTlZbjJvRFpkeHJlUHV2OFk3L2g1TG9wZGdKQWtaODRnRHBiZjExZEpKbDRlZWh
4OENZWFJ5MkcvejM3dUV6QnVWTHo2U3JlQzlmbGRmRXdyckFaR2YyTzVRdmZCTkFEMVFaMUhaYTQvamFueCtMUzNmVjI5dFRQa29OSDJUOWlWSWE2TW
pOYVhlYTU0dlYveng4c3JwNkxZaC9DZUF1THg3M3pFMHFQUUdFSlNmZEJBaGpNdUNveTNoemo5U3d6VGl0QXpwS0VsR2lBZXh3OHV1U25SSXBxb0ZGN
jZ2d3k0KzczOVQ1VElQSnhkUmQ0dnErcXBKU01pNU5IbzZDZk5TVnpBeHVLbjVCTW81SXF4aFdrT1d6ZitKSFE1dkVLQzIxamNpZzVOcGI5U1JrUDJL
T2xqRDVTbzZ2UEJkZTE3V2puRE5nYktTODBMQjVrK3ZkU2pxYVVGbGdMMmV4SWRBUXEwbE52ZENxRVVhS2QwdlRHL1dvOXU2RWVnMGNveHZwbGNMbW5
oY1cwSS9KWTZhUWU0VE41NC9lZ0lvYkFFVVdhMTZ5ZjRTU0IvbzVCNXJYOCtJdnhmcTZaT3Y1QmdTRkc1Ky9aTlQ4NHhMQkJiQzZhcjNERDFrNTg1UV
hOcmk3cjRLL1ZRWHUvdmR2UHUxUVhwT2xDcGxWK0xkV2dVZG5odWlmTE45Q0VzWlRxelFYSXpUNzhaK2dQREpoVWJSWi9VQWo3UCtQYmxmaDhtZHhrS
Gc4TUlBcENNJaFNPVUJ0U1RVQ00rK3d0OWswOHIxcUl0OTZFQ0F3RUFBYU5qTUdFd0h3WURWUjBqQkJnd0ZvQVVoVDJnNGZ5eENTZmsycjBmaG90QUJu
SkNDb0V3SFFZRFZSME9CQllFRklVOW9PSDhzUWtuNU5xOUg0YUxRQVp5UWdxQk1BNEdBMVVkRHdFQi93UUVBd0lCQmpBUEJnTlZIUk1CQWY4RUJUQUR
BUUgvTUEwR0NTcUdTSWIzRFFFQkRRVUFBNElDQVFBVHREQzBYb0gvWTZyZ1RVRHBxUHIyREtvZ3JsWHZjcTZaL1ZxajEWk5RTGMxREp5UHdDSkRVbW
tKaEIyOVd1YmE2cjlGZkFHdDRjS0hwMnJaNXFYYWdYMnNOSWdEdTZ0d3pBeXFCeC9JM2JUbWhTYWpQemJvVlZyWlpwaGp1dTYwa0RsaGZZVTZDOGxJO
VdqVHpjVGNyYlpOd2VlbUpzZVpmOHJnb2V6YXlPemZNZk1nVjNFbkdIRW1BRUZ5T0Q1bzlSZkg0SnMwVXdxaHdhUWVkL2syTmNyTzZPS3VwSmp4cE5p
cDdWMS8rM01BNWZsc2k1cFM5Vmg0ZWJRdXFLRzhjWXJhWHB3QzJKWXgyblZCU3ZDTVFvQTBCRks4MWlVVXVDeWdCZGwzaUE3RFFCNzJ4Uk1KUzFVSlF
IemZkRDJoWTlWVzJHeU5JOXBNb2NBT1JvWFF1ZWdESitpbFh2WGRzOUhMd2lqUFV3TzR5bVpxZUFnSTZlUVlMQXNjbU50MTNwSjJOYkZYVXhhRW5rVl
grWUV5UDU0b1Z5KzlwQ1EwN1B0ZnJNWWZCbWN4ZGZmcHFOaEM3N1IzTmpobk5jeEVKSVdvUHFjU1hwcFNNRVQ0dktvU2JiZWNrV2xkSFpDVHdrRVhSN
1VIYmxiNG5rTllkUjhZRmZDSDhjV3VHUlBvUnRoM0ZSeHZhalJPSGRjRUZtRkJnUGtjRG1kaU0rbWZJQWRUeWdJQkhnenU5MDFJWlExRWErbzVPU0o5
L2x3RWpCZ3VXdnhQU3VKS1hqZG1ta3ZYUjQ3UzlpcnR6RVpjUHdUWUFML3JKM3VyZmIwTE0yeGoxSEx3aUQwSGJMbmRkZnhMRHR5dWJmVHN5ZzRUV1F
0dVJwb3ovTU9hY1RQb242RC9MQmlOOVE9PSIgXSwgInN0YXR1cyI6ICJvayIsICJyZWFzb25Db2RlIjogIjIwMCIsICJyZWFzb25UZXh0IjogIlRyYW
5zYWN0aW9uIHN0YXJ0ZWQiIH0.MEUCIFRKxzIo9qf117hXxQdMgsMz7wGWi7PUz8sm5NquDJrbAiEAu-E5ynZHNrHD5KzVFwyQ2poMdZyt5m-
QkDbeYoPBn10
```

JWS Header

```
{
  "alg": "ES256",
  "kid": "6173DB6EE68DA64F26435F255E33C2E1B5956ECA",
  "typ": "JWT"
}
```

JWS Payload

```
{
  "version": "1.2",
  "transaction": "MFkwEwY...
```

```
  "chain": [
    "MIIHKzCCBROgAwIBAgIE...",
    "MIIHezCCBWOgAwIBAgID...",
    "MIIGEjCCA/qgAwIDA1tg..."
  ],
  "status": "ok",
  "reasonCode": "200",
  "reasonText": "Transaction started"
}
```

2.7.2 Execute transaction

Execute transaction and SCS response messages are encrypted JWE tokens. Used encryption algorithm is AES-GCM.

- Encryption key is the result of ECDH key exchange algorithm. If the key exchange result exceeds 32 bytes, only leftmost bytes are used.
- Key management mode is “Direct Encryption”

Encrypted signature request payload follows chapter 2.6.1 definition with the difference that ***selector*** is not present.

Example request:

```
POST /Sign HTTP/1.1
Host: localhost:53952
Connection: keep-alive
Content-Length: 350
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45
Safari/537.36
sec-ch-ua-platform: "Windows"
Content-type: application/jwt
Accept: */*
Origin: https://127.0.0.1:53952
Referer: https://127.0.0.1:53952/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,fi;q=0.8

eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdDTSIgfQ..UdIVcZ_vwoW8Po7
2.K4Ob5TTmJdAxAgmph2YRtxAekVlowUOpaAqEjduRWFo1q_6bTozqgDUIzCw3k
tR9dP_343uCp8FoopgOxlxpEo6DpBjHVp33CVSnb11EbLzGvaJGWJVMD_o94He0
z3cvpzY8ZTkNi9Zk1RQFo-vWpmU-7wGb0ZM1z-
MnvvY4AzmxiqMIRyu5urW3MGKWH8skBHJiaotJUEqi07T-
voiDEpC_U7Zp9a_cSGolJJPdB57v587NI6wdvONdA7Ce2oG7v2HdRw.DpS-
ylmtrvDHI4b5WHsX9A
```

Example request details and test vector for developers:

JWE element	Value
Header	eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdDTSIgfQ
Decoded header	{ "alg": "dir", "enc": "A256GCM" }
Encryption key	ECDH key exchange result: 658F205270C4C5C6C7217A039870C0337DEA1F41662DFF5 7AC3A8BA7DDFCBC8B
Initialization Vector	UdIVcZ_vwoW8Po72
Payload	K4Ob5TTmJdAxAgmph2YRtxAekVlowUOpaAqEjduRWFo1q_ 6bTozqgDUIzCw3ktR9dP_343uCp8FoopgOxlxpEo6DpBjHVp3 3CVSnb11EbLzGvaJGWJVMD_o94He0z3cvpzY8ZTkNi9Zk1R

	QFo-vWpmU-7wGb0ZM1z-MnvvY4AzmxiqMIRyu5urW3MGKWH8skBHJiaotJUEqi07T-voiDEpC_U7Zp9a_cSGolJJPdB57v587NI6wdvONdA7Ce2oG7v2HdRw
Authentication tag	DpS-ylmtrvDHI4b5WHsX9A
Decrypted payload	7B202276657273696F6E223A2022312E32222C2022636F6E74656E74223A202266344F785A582F782F464F354C634742534B48575866777453782B6A316E636F5374335341424A746B476B3D222C2022646973706C617954657874223A20225472616E73616374696F6E2074657374222C2022636F6E74656E7454797065223A2022646967657374222C202268617368416C676F726974686D223A2022534841323536222C20227369676E617475726554797065223A2022636D732D706164657322207D
Plaintext payload	{ "version": "1.2", "content": "f4OxZX/x/FO5LcGBSKHWXfwtSx+j1ncoSt3SABJtkGk=", "displayText": "Transaction test", "contentType": "digest", "hashAlgorithm": "SHA256", "signatureType": "cms-pades" }

Response:

```
HTTP/1.1 200 OK
Content-Length: 3991
Content-Type: application/json
Server: SCS 1.2 (2021-12-01)
Access-Control-Allow-Origin: *
Accept: application/json, */*

eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdDTSIgfQ..3-
ceXbhoZ6ReqFGF.3VgpdZZAdJrP7EXt691QAjTthPM8kxcGcvqxlmPQPopE5tf0hMPMLLKwut_CIH2iHZ17ke2oTg3_SiCyvfnfRgEuk83qCO2QYwn9
ZnX1dnazR02RuU_FplSXdMiRRk2cUWZb8NoJ_Y38QufO7TKCOlfEfgi1dF7KxC3EXUurEy1wZ4wDPma49wNHiwZfLXxlNRjUuja96VNwQ4yfC8hriHg
BluuKaRATd5AqGBW-7ApC6TPdm7xukzQfpiOvG3nRgUgqpeVcEwquSW5lLfX-VQcR-9L2jwrtL88rTrlIXZgyBN7psQmbqiIuYqAZY2S-F2c7-
O7VCF_fmpHzxPm0QST0N7CV2_fZJLMh6U_O-
xeTyPd7HbBuTxZ8rtWhwTDdvarIMofrQAt2j__yKNgtIZT9LiDzYvL1u27tQskjY8ghK82NiwZBtGdOT9zkM2MBnYmTs8SFAOyxhjUSM5G553-
IARat-YBtXR_46AOd6uDbZ8nWbm-dtWIpab02yd_7CtEk6CSAEzQO7wEOa8fnNwdEKROgNR5EAMnoF9Qs-
iwnvyLcBm262lpQ8Mo71LMJA4Wbys9q8QZLWKqO-
a79qLO1kH_zV7QL_JBSsWJcj7DvhNbOz2crOmriiGcn5XPSs8VhmCm_kSlps7cLsGFBnwVKvQSWBL3nLao7Mn7Rr-EndLHRcgI3pHU_5sZj-
SaBqbpgbaHJjiYGV2fTMh9P_Zt4dfoTyIeZiv_2pxB-
EnUL4F6hyUCo0Tqz0efRqba0Ho3yXa1GwOUaC7CJx2sKhS8MBNCR_EalDoGzF4tUJNtMyRxMT6daiwdrFznxgb-
noqsob8w4mowZ_6gmGAIRLB2CrJH16QhOzt_R1Y0sv-Ad4agfikXWtKITWUC9wUxo6pNzlWUSE8Mte9S0SF13iJjsyFH9ms87KaYYU_GsHclKn8JPL-
GeKh-1On_n2ELBUlH3_n6PX6q1lbXa-vTqtw0y8ee3GBKWFUM0WJLKTH-
1qNpYS64UJGUHzbGqCNGa20YBLJCeF7O3DIEB6zYMxoYQIdbHaN5y0a4zffFyqXQE8zcgXP7AZ6G7QbPpmKw1JiixowaX7hWpx3b_0kDKKan9M7tZKx
qSZ3L8tVg_soHNzdDIr2sLYWVp07YKpglO2ooMCYlp3OsH6hOXfngme8Xt_cdSY00ZlZDcoHIHGdFOnUJ6HoSBgGI4nwYMA8d5lSvVYVb7_vVGkqhFX
bHtd15xAT6u-WuBYJ0DlsXyWdhPDUKZR0UEUISNyOn-ZGBA3aeuAG81NpRLaFk6xqeXuz5QQinRu9knvQaIymFHQ5p4GCexYg767LvNOUW2W-
_J5dKUy1lknB36tyAQFz-g4qOEzvo2drX6nfU4Zv-Xvy_4GsVKRcBrlIRz4n9sTGd0Eix-IBTvTpL3-6vtw7UIdQrz-GS7pdFj-
R2PNWlZ758Sqa06oQikeU1bqNcDfN-VIDk09iomETDRlo4j003QaUy-VmIZ6OIG-
Ng208zUOgk6v_lfVIiCckStXhptWFoikstWo3fAcKcNOuFxIXE2QqHYhr8itEMTiiRe0eu5PxsyVOMoVaBwz-
qbuioD1lU7PpOvq6L995q4s71WssHWfR3jK9JMHVuTp6f5p2Ol7UW1Gil7MtHSsmjOx_F3DoRWnxQe8NqCuVSAo6NOYq5W323kXLpTc0FD143TwDPh_
AZT9zPXiB0_uKYvmwF0B05Ds28DnJyJryr06A4lM3IO_ORL-fgCChvfmzHqKfTsu_i5-
ieVwrtXGMHwEmi4LtsQw83_xbPmejDeYwcjxH5vrtTYBn2lkysfGU-KBM2MDWxys8ZZdTi99t6xVFC9k3rfdpDK2ryqjdysHzY1mUtso5rhDf-
fxMyGNC94i5T6Skh98F_7p-qtpLoik_bGDUqEH51n5HxHANogBISFR2nqdm_yAfmzjOVzqheNMJPN5BuyTCB6XNE9Hm0P0SFxu35MAFZROW-
OqHpyMWnaiJ4XNUiSMOVA0SdNmo73z0wQbcCucMyxmiHzMFW0S2ZackSBJDcbELnMPupi7rfnpB1AhjmHc_Wb-k8yFTyg8M46ND-
AvIuXY0coDo5jPJSuhaK1eBxbt0zyxxFOiNwWbEvVDFsMkfsf5EeMy9i1E6lZMeItf7oRNpJGMwqcXX-
Cz7jFLExIdZb8AJ8ry3cEW8trWzqFr6ixnIXWm_S5D2fiYr2QeFsTCvE816ukuErIee-S1_CaU81FX4r3d3JaxRDoXNG-
DOBB6kRvn2Q3DY2YUY2gBkTz7Kdco9a_B4gS9_M1tv3hD1XLBO0R_SOAEcB_qrQZWRoR5r8_krsLA8gPR5IaOqTd6cqB9Ayxu23z9-
PTLmDelpdsyuIOaEjkeuW9iA8MOZeAlp3kLcoehHjLc7qrdPho4fBhvbVWa-ckSZYyEd8S7YZwjGCdbFmjbcHIJ66GDNnBFdraDsba11ztdSLpv-
CkQJKCvRor2s3nYWqIYCcdE5C20hNW4AoXESUcYHCuMtJQINKLo686l90T_ixuquTYgcYY6sCO2kQEwfsY_0uVPDzIWlIPa43VHeGa6VHOwodOjZ_tW
OWnnKqIFVHRzytYdhN3U4WtOCwj9QIi6biMi0dtgJRmXYgoSwz724-
bx2KXjveG9lsHG4QqB_6H11DKtsfsLOzd5tQYjOaWgltDq1aZu1VD3DXpPboVinVCkNp4G-FHyupyCnUMuZD-uKZePU0ipl-f0jbEZpY-
mFVnU8Y65mD-7b-ID-njydxymXMKLLtb8P_I5Ksl2LC3YeeNW_1SUUY4aEtZc8IzxIGxLlNKmISfcW-
iSfgKJMuWiffcjZedmrrj49rsWOdY1C1xyjyNcY3_b4IQdGA4cydh8AZZDOifAfVVE8nWNoqmZDvZTWdfjbWVHZLmjDCEAnGz9UYB0Ews5m6ZiBIjSQ
YkfIVlyvOPLoeqHG-ZqPOr2JLek5MuXg9hnKg8r_woih4_bTstksIUWD7MBIupx1EGaGtEplCvObKHerxEnPTbLsmjz_CelBMJ-
qtaWG5l_gC_ftoDiFifRfFQbdrj5_BOO-ROcyiD33K-929gyrH3q5napyk-ibTAP-
kOWmEtTjsebnrFWSfSeKAN4GN0DgkFJ24GwYnORmxsZyTOdFBU2w8yO85hbVZXzEzX6EVtAVi30HCsn8ev1pCDWCCXpIlw8IxiWlUcuFNCAO8evdC4H
7LBx3rEIxzPOoESJrOSh-
L9lKwr4lYJaFihEC9uE5eZzvhePFVCoBRMoaqJK815W6fCJ7OhLV0CXS3eaiUHthvHakZ9VWlg2_BwB06E2htPkyQ0H49WqQCzLbTVrYjPIuACaNGkM
WCG6UfoawzMVYsn_RmRrHIco0LVhAVr9uJOFhnPcOWt5dqeX7ujZ7gxFiFXLRxYRe3ClE-
nz8aHaXXbCiBduTbanurpMoUUjVh4C2rvduajK5vG7k_aKHmUKZKGK0eAlFdSY_tQqwdB4fIu9RixhNsJhsMBh_FDxR4jZMM0Kw6u89hv0OQKQLXSVX
x98Y_SDn1PUb9NiV2JNweHtAPfBAErFTZyAtn9t0SGA9Drx7Mx972a85NloTDECMQnjBL74jiGgs_Hii0_pOK1W79pqHIx8d1T83sXTBLeM_MhBfR53
zmLZLJ8PIym_M8NypgOQoezS0nJhtgzz_SJ_A3EEnRvSQ210hBSqWO7xnXtN64ZxBbfqBJyCW9qcYWM5gR3h7ImTnpqz3nIX6POusBDp7QVLmdfSLKD
fdSguWwpAm6A7e7rCt__cwrAqPdUjtsCa9fE4z7t5ppv5UexA2MwvcGYdDls47uQ3O3NWbFFO2A-
RJTTQwn00y0A41soZXk4YNhZSUb5we9S_sNSi1N5z2G6g1J921JTucJ5hifADuPsBoQ2qBmhwYdvmhI2_6MqEQwQCoJa2cW-
```

```
nWIOra2dqMbUyTPv8C3sQgkckIpEOE0PjmkyL4WUztLe5MSFw7yBQcQ-BCNLYxvti6mKOsmA5iZ__JqFVKmK3lG3nDMP8puHvyDFCXNEL3gLTl2ju-
1Dyf2zvv7TPyZs8LKgiV2gtOgw38s_EfBXdiez-DWobTGAIMmkUYIjsWWQJwqkOQPLvfPL9KLmKjZlcCmE0O0mk6Nh4qRcUbHXPRkA-
Gdas4WFtfl3MQ3AaHyxswQCXTfzE0Yga5Our7qQf2w9vbw1woNAemk8SeZhDIyhgZ1mjulQ9Iz5eeRq0aff2sedsehZBOBH6dtEx0DaGYQe9t9uu1pL
eEnTAd8zrpoDjZq4pUd2EIpOx3EptDy96U-Vxo8aQVLy5oK1wThJ-
qBMqkJS_ehak57cUbtR_MXBmCEK5N6HbrmxZcJ_FUEKHrDXO9LWMmpnafZPA0kaEyDWEM5ZzJ5AUt8N8aX7-
s8ft87qdZLA8LnLSjdm4iP1QKV_1pfRJ8cNgFDxR4hFczJ5mBNfH5sGKhJ65AU3BIi8_CeslYw7gEHRCXLi1_-
j7m6dJyH397uDcsebW2Npf3j0v8VrrVnwgTJ1S2xzleNbQcmqjCXdeDbxwt9VxBSFklFYdpgd_wk8l536LyMpe3x0iyWbzZ2NZwRnd6z0zq8ALDBK7S
_XDXWyVMNc94MrDGzCdzGjVy_ZaDD_uhsnSScxiGN00L5FnUSkqMMC4Xnd80amNIX8e6aO8WOLqeuLKjulSEkl8QLIt7ZAOeD_Rf8RtPuNwDegMefhn
qbdDpIYLXDFD--q4qn1H8iwfoutHHN5ZxiL_JmsfmNAXfemsrDw4qgsWCihPTJPBmlet-
UY31UrLaR6oqaFpmJgZ2dlm9Pj8GIkvM59zEHN8zNwcV7skG3dm73NWzM0ng3oQqOkIazmkWXw8Er23MjqzY0AISYxcte94elLexs3uoMOb0k9DaA39
7tKxH3YNBLkY9gix6v5Vwjvpp_bSGSEJsRdeC1IFDsavr8VBKAJix-eIU61mboEgdepLZEDJgkv3h5AS0u2B5-
eVaFCUEIeLoRslYCJZM3p5G_VZTCwHCZH7NC4Dl7vJ6kzTPhUrsEydjA-
87jCgBQ567jW_whiwIOUKIEG0xseSJQ3BrnWNfl3e63NzRCg0yp4niducBaS9pTr8Pmp86A.FFUjIAH_HJEZq7XRuXmV6g
```


Response details and test vector for developers:

JWE element	Value
Header	eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdDTSIgfQ
Decoded header	{ "alg": "dir", "enc": "A256GCM" }
Encryption key	ECDH key exchange result: 658F205270C4C5C6C7217A039870C0337DEA1F41662DFF57AC3A8BA7DDFCBC8B
Initialization Vector	3-ceXbhoZ6ReqFGF
Payload	3VgpdZZAdJrP7EXt691QAjTthPM8kxcGcvqxlmPQPopE5tf0hMPMLLKwut_CIH2iHZ17ke2oTg3_SiCyvfnfRgEuk83qCO2QYwn9ZnX1dnazR02RuU_FplSXdMiRRk2cUWZb8NoJ_Y38QufO7TKCOlfEfgi1dF7KxC3EXUurEy1wZ4wDPma49wNHiwZfLXxlNRjUuja96VNwQ4yfC8hriHgBluuKaRATd5AqGBW-7ApC6TPdm7xukzQfpiOvG3nRgUgqpeVcEwquSW5lLfX-VQcR-9L2jwrtL88rTrlIXZgyBN7psQmbqiIuYqAZY2S-F2c7-O7VCF_fmpHzxPm0QST0N7CV2_fZJLMh6U_O-xeTyPd7HbBuTxZ8rtWhwTDdvarIMofrQAt2j__yKNgtIZT9LiDzYvL1u27tQskjY8ghK82NiwZBtGdOT9zkM2MBnYmTs8SFAOyxhjUSM5G553-IARat-YBtXR_46AOd6uDbZ8nWbm-dtWIpab02yd_7CtEk6CSAEzQO7wEOa8fnNwdEKROgNR5EAMnoF9Qs-iwnvyLcBm262lpQ8Mo71LMJA4Wbys9q8QZLWKqO-a79qLO1kH_zV7QL_JBSsWJcj7DvhNbOz2crOmriiGcn5XPSs8VhmCm_kSlps7cLsGFBnwVKvQSWBL3nLao7Mn7Rr-EndLHRcgI3pHU_5sZj-SaBqbpgbaHJjiYGV2fTMh9P_Zt4dfoTyIeZiv_2pxB-EnUL4F6hyUCo0Tqz0efRqba0Ho3yXa1GwOUaC7CJx2sKhS8MBNCR_EalDoGzF4tUJNtMyRxMT6daiwdrFznxgb-noqsob8w4mowZ_6gmGAIRLB2CrJH16QhOzt_R1Y0sv-Ad4agfikXWtKITWUC9wUxo6pNzlWUSE8Mte9S0SF13iJjsyFH9ms87KaYYU_GsHclKn8JPL-GeKh-1On_n2ELBUlH3_n6PX6q1lbXa-vTqtw0y8ee3GBKWFUM0WJLKTH-1qNpYS64UJGUHzbGqCNGa20YBLJCeF7O3DIEB6zYMxoYQIdbHaN5y0a4zffFyqXQE8zcgXP7AZ6G7QbPpmKw1JiixowaX7hWpx3b_0kDKKan9M7tZKxqSZ3L8tVg_soHNzdDIr2sLYWVp07YKpglO2ooMCYlp3OsH6hOXfngme8Xt_cdSY00ZlZDcoHIHGdFOnUJ6HoSBgGI4nwYMA8d5lSvVYVb7_vVGkqhFXbHtd15xAT6u-WuBYJ0DlsXyWdhPDUKZR0UEUISNyOn-ZGBA3aeuAG81NpRLaFk6xqeXuz5QQinRu9knvQaIymFHQ5p4GCexYg767LvNOUW2W-_J5dKUy1lknB36tyAQFz-g4qOEzvo2drX6nfU4Zv-Xvy_4GsVKRcBrIIRz4n9sTGd0Eix-IBTvTpL3-6vtw7UIdQrz-GS7pdFj-R2PNWlZ758Sqa06oQikeU1bqNcDfN-VIDk09iomETDR1o4j003QaUy-VmIZ6OIG-Ng208zUOgk6v_lfVIiCckStXhptWFoikstWo3fAcKcNOuFxIXE2QqHYhr8itEMTiiRe0eu5PxsyVOMoVaBwz-qbuioD1lU7PpOvq6L995q4s7lWssHWfR3jK9JMHVuTp6f5p2Ol7UW1GiI7MtHSsmjOx_F3DoRWnxQe8NqCuVSAo6NOYq5W323kXLpTc0FD143TwDPh_AZT9zPXiB0_uKYvmwF0B05Ds28DnJyJryr06A4lM3IO_ORL-fgCChvfmzHqKfTsu_i5-ieVwrtXGMHwEmi4LtsQw83_xbPmejDeYwcjxH5vrtTYBn2lkysfGU-KBM2MDWxys8ZZdTi99t6xVFC9k3rfdpDK2ryqjdysHzY1mUtso5rhDf-fxMyGNC94i5T6Skh98F_7p-qtpLoik_bGDUqEH51n5HxHANogBISFR2nqdm_yAfmzjOVzqheNMJPN5BuyTCB6XNE9Hm0P0SFxu35MAFZROW-OqHpyMWnaiJ4XNUiSMOVA0SdNmo73z0wQbcCucMyxmiHzMFW0S2ZackSBJDcbELnMPupi7rfnpB1AhjmHc_Wb-k8yFTyg8M46ND-AvIuXY0coDo5jPJSuhaK1eBxbt0zyxxFOiNwWbEvVDFsMkfsf5EeMy9i1E6lZMeItf7oRNpJGMwqcXX-Cz7jFLExIdZb8AJ8ry3cEW8trWzqFr6ixnIXWm_S5D2fiYr2QeFsTCvE816ukuErIee-S1_CaU81FX4r3d3JaxRDoXNG-DOBB6kRvn2Q3DY2YUY2gBkTz7Kdco9a_B4gS9_M1tv3hD1XLBO0R_SOAEcB_qrQZWRoR5r8_krsLA8gPR5IaOqTd6cqB9Ayxu23z9-PTLmDelpdsyuIOaEjkeuW9iA8MOZeAlp3kLcoehHjLc7qrdPho4fBhvbVWa-ckSZYyEd8S7YZwjGCdbFmjbcHIJ66GDNnBFdraDsba11ztdSLpv-CkQJKCvRor2s3nYWqIYCcdE5C20hNW4AoXESUcYHCuMtJQINKLo686l90T_ixuquTYgcYY6sCO2kQEwfsY_0uVPDzIWlIPa43VHeGa6VHOwodOjZ_tWOWnnKqIFVHRzytYdhN3U4WtOCwj9QIi6biMi0dtgJRmXYgoSwz724-bx2KXjveG9lsHG4QqB_6H11DKtsfsLOzd5tQYjOaWgltDq1aZu1VD3DXpPboVinVCkNp4G-FHyupyCnUMuZD-uKZePU0ipl-f0jbEZpY-mFVnU8Y65mD-7b-ID-njydxymXMKLLtb8P_I5Ksl2LC3YeeNW_1SUUY4aEtZc8IzxIGxLlNKmISfcW-iSfgKJMuWiffcjZedmrrj49rsWOdY1C1xyjyNcY3_b4IQdGA4cydh8AZZDOifAfVVE8nWNoqmZDvZTWdfjbWVHZLmjDCEAnGz9UYB0Ews5m6ZiBIjSQYkfIVlyvOPLoeqHG-ZqPOr2JLek5MuXg9hnKg8r_woih4_bTstksIUWD7MBIupx1EGaGtEplCvObKHerxEnPTbLsmjz_CelBMJ-qtaWG5l_gC_ftoDiFifRfFQbdrj5_BOO-ROcyiD33K-929gyrH3q5napyk-ibTAP-kOWmEtTjsebnrFWSfSeKAN4GN0DgkFJ24GwYnORmxsZyTOdFBU2w8yO85hbVZXzEzX6EVtAVi30HCsn8ev1pCDWCCXpIlw8IxiWlUcuFNCAO8evdC4H7LBx3rEIxzPOoESJrOSh-L9lKwr4lYJaFihEC9uE5eZZvhePFVCoBRMoaqJK815W6fCJ7OhLV0CXS3eaiUHthvHakZ9VWlg2_BwB06E2htPkyQ0H49WqQCzLbTVrYjPIuACaNGkMWCG6UfoawzMVYsn_RmRrHIco0LVhAVr9uJOFhnPcOWt5dqeX7ujZ7gxFiFXLRxYRe3CIE-nz8aHaXXbCiBduTbanurpMoUUjVh4C2rvduajK5vG7k_aKHmUKZKGK0eAIFdSY_tQqwdB4fIu9RixhNsJhsMBh_FDxR4jZMM0Kw6u89hv0OQKQLXSVXx98Y_SDn1PUb9NiV2JNweHtAPfBAErFTZyAtn9t0SGA9Drx7Mx972a85NloTDECMQnjBL74jiGgs_Hii0_pOK1W79pqHIx8d1T83sXTBLeM_MhBfR53zmLZLJ8Plym_M8NypgOQoezS0nJhtgzz_SJ_A3EEnRvSQ210hBSqWO7xnXtN64ZxBbfqBJyCW9qcYWM5gR3h7ImTnpqz3nIX6POusBDp7QVLmdfSLKDfdSguWwpAm6A7e7rCt__cwrAqPdUjtsCa9fE4z7t5ppv5UexA2MwvcGYdDls47uQ3O3NWbFFO2A-RJTTQwn00y0A41soZXk4YNhZSUb5we9S_sNSi1N5z2G6g1J921JTucJ5hifADuPsBoQ2qBmhwYdvmhI2_6MqEQwQCoJa2cW-nWIOra2dqMbUyTPv8C3sQgkckIpEOE0PjmkyL4WUztLe5MSFw7yBQcQ-BCNLYxvti6mKOsmA5iZ__JqFVKmK3lG3nDMP8puHvyDFCXNEL3gLTl2ju-1Dyf2zvv7TPyZs8LKgiV2gtOgw38s_EfBXdiez-DWobTGAIMmkUYIjsWWQJwqkOQPLvfPL9KLmKjZlcCmE0O0mk6Nh4qRcUbHXPRkA-Gdas4WFtfl3MQ3AaHyxswQCXTfzE0Yga5Our7qQf2w9vbw1woNAemk8SeZhDIyhgZ1mjulQ9Iz5eeRq0aff2sedsehZBOBH6dtEx0DaGYQe9t9uu1pLeEnTAd8zrpoDjZq4pUd2EIpOx3EptDy96U-Vxo8aQVLy5oK1wThJ-qBMqkJS_ehak57cUbtR_MXBmCEK5N6HbrmxZcJ_FUEKHrDXO9LWMmpnafZPA0kaEyDWEM5ZzJ5AUt8N8aX7-s8ft87qdZLA8LnLSjdm4iP1QKV_1pfRJ8cNgFDxR4hFczJ5mBNfH5sGKhJ65AU3BIi8_CeslYw7gEHRCXLi1_-j7m6dJyH397uDcsebW2Npf3j0v8VrrVnwgTJ1S2xzleNbQcmqjCXdeDbxwt9VxBSFklFYdpgd_wk8l536LyMpe3x0iyWbzZ2NZwRnd6z0zq8ALDBK7S_XDXWyVMNc94MrDGzCdzGjVy_ZaDD_uhsnSScxiGN00L5FnUSkqMMC4Xnd80amNIX8e6aO8WOLqeuLKjulSEkl8QLIt7ZAOeD_Rf8RtPuNwDegMefhnqbdDpIYLXDFD--q4qn1H8iwfoutHHN5ZxiL_JmsfmNAXfemsrDw4qgsWCihPTJPBmlet-UY31UrLaR6oqaFpmJgZ2dlm9Pj8GIkvM59zEHN8zNwcV7skG3dm73NWzM0ng3oQqOkIazmkWXw8Er23MjqzY0AISYxcte94elLexs3uoMOb0k9DaA397tKxH3YNBLkY9gix6v5Vwjvpp_bSGSEJsRdeC1IFDsavr8VBKAJix-eIU61mboEgdepLZEDJgkv3h5AS0u2B5-eVaFCUEIeLoRslYCJZM3p5G_VZTCwHCZH7NC4Dl7vJ6kzTPhUrsEydjA-87jCgBQ567jW_whiwIOUKIEG0xseSJQ3BrnWNfl3e63NzRCg0yp4niducBaS9pTr8Pmp86A
Authentication tag	FFUjIAH_HJEZq7XRuXmV6g
Decrypted payload	7B202276657273696F6E223A2022312E32222C20227369676E6174757265416C676F726974686D223A20225348413235367769746852534122C202273696 76E61747572655479706522 3A2022636D732D7061646573222C20227369676E6174757265223A20224D49494B7567594A4B6F5A496876634E415163436F49494B717A43434371634341514578447A414E42676C67686B67425A514D45416745464144414C42676B71686B6947397730424277476767676763764D4949484B7A434342524F6741774942416749 4542677A476C44414E42676B71686B69473977304241513046414442354D5173774351594465515147457 74A475354456A4D434547413155454364377761566D466C63335276636D567261584E305A584A706132567A6133567A4946524655315178474441574 2674E564241734D4431526C63335270646D4679625756756447566C644445724D436B4741315545417777695 66C4A4C494652465531516751304567 5A6D39794946526C633351675548567963334 7397A5A584D674C5342484E444165 5467730784F5441314D 6A41784D5449344D6A4E61467730794E4441794D6A55794D5455354E546C614D483078437A414A42674E5642415954416B5A4A4D513477444159445651514B4441565 564577873615445534D424147413155454 2524D4A4F546B354D4451354E7A45774D5134774441594456515171 44415645 5255354A557A45554D42494741 3155454 24177 4C536C565656456C4D51556C4F525534784A44416942674E5642414D4D47307056565 6524A5445464A546B564F49455246546B6C5449446B354F5441304F5463784D4 44343416149774451594A4B6F5A496876634E4151454 2425141446767475041444343 41596F4367674742414D68594D305533737A557669446F6E2F6C4C71447567484C6F734F6363733868686 2463877694B52527A2F4C487056593459733 37649756D32636334667A31684C673330686B5A4435305878696371483932574356 6B6D7756366E5 86569366E4A7A684E4975374179582B666E5A78674843527064796B63787555676378426369346C586E4470706B6455624A7662764635526 1586B6C4A78706E6B67435 56E4D4B3261535158436D4368302F6E5366444471336F6342454268704A387A512B7253434764714762757065674777454E33637A4858624C4E724F397166 31556C3164 6741736A34436B30504B394F69636A366E7867474C62666A39663879446E45566649614761506961544239352B3146416C6554734D543859535941633237437 74663486 864643031704C7162386A734468394D41733168393271746D4C474365325877546B34716E3139346163444377 6E6765654F79776B526241786C476776376D 586D6C313 3692F45486A6A7A33536F5949546F506665563749596E46497975357A747630437839675372694370 6F774D2F412B523541493043565550784 1524D33465063766E3039 7077327677456355 6B4E36756A6442506F345A35792B4851626B5678773172435379776357477155715030516854564D64344F6972635A456876516479334 F4E66574547 796371543248386A66375A70773556315149444151 41426F3449434E544343416A457748775944565 2306A42426777466F4155505A716A746667564 5653852797 5764858 453254674C4C4850384577485159445652304F42425945464C466B77784C6333657945653862335131367 073594C62705435654D41344741315564447774542 2F77514541 774947514443427A6759445652306742 4948474D4948444D49484142676F716758614542574D4B675573424D4947784D436347434373 7347415155464277494 24668746F64 4852774F6938766433643 34C6D5A70626D56705A43356D6153396A63484D354F5338776759555474343734 7415155464277494 34D486B6164315A6863 6D316C626D356C 634739736158527061577472595342766269427A5957463059585A706247778684943306751325679 64476C6D615774686443427 76232787059336B675A6D6C75626E4D6 74C5342445A584A3061575A70593246305A534277623278705933 6B6761584D6759585A68615778686D786C494768306448413633 6D616D6C 793933643636 3755A6D6C755A576 C6B4C6D5A704C324E77637A6B354D435547413155644551 51654D42794247 6D526C626D6C7A4C6D703164585 2706247 4670626D5675514852316247 78704C 6D5A704 D413847413155644457745422F7751464D414D42415141 774E775944565 23066424441774C6A41736F4371674B49596D614852306 3446F764C334279623368354C6D5A7 0626D56705A43356D6153396A636D7776646E4A72644841305979356A636D777762675949 4B77594 24251554841514 5455 96A4267 4D44414743437 3474151554 6427A41 436869526F644852774F69387663484A7665486B755A6D6C755A576C6B4C6D5A704C324E684C335A796133 52774E43356A636E51774C4159494B7759424251554 84D4 1474749476830644841364C79397659334E776447567A6443356D6157356C615751755A6D6B76646E4A72644841304D4330474343734741515155464277454 44243457748 7A41494267594541493547415 1457745775947424143 4F526745474D416B47427751416A6B5942426745774451594A4B6F5A496876634E4151454 E4251414467674942 41 44372B574161763357664459324E335749695874677053776D452F614859534842736B552B2B6D4B35556B33423164453 45747612B6B753168713357 764B4E513650747 4477137576347577064553453563868705943514B63544A657659596F6339576B4E345A4436456E63464A2F7A5330302F634F5A52516C774978646F35456D7959696357 79414E664B6A3139504C6C566236454C714B524731766C45496970434A7252674650634A6A2B43765773785A3164384E30326E4B616A6D47704734343378374A376F4 C78584E574F6742533652735532363243646E394F4B6B794B70673569586970345079677 950494A3663636F30705977442B327A7170626D7577444 732514C474769 2B59 355435314332764C76676C466C4B4B712B324B426D396652582F6568474E3853686A37656F76314D6550414537766E744D436134465932683968 375532362 B5A366352 51697559496C4D2B7871696E787A302F4C696B64527075344744472363262514778503542343 24F5579536C797A4D65514C73656F4C374E484F 7472344B525661464434 717036355049797135325 3516F745A6C544852715353684 6506F456E3462665750396F68344E47526B30754B63464C724974 457069787551722B386559447 06D6B31644 D493072733432516 56C543067427832413 92F61554736765671715A7A704F4E7A5A4C4F4A394A45486C6D4153752B522F50535074704C314B736179775A7570714A57 7847594E7674697658725 37A4E4B35522F696B795A4E6B5A3069445274716 2466B6B44386A704F6D4935613278646 37A49306637303772784A4C695838 5444354 3632F 376A7639362B7A71536A736D6A327A73516B6974666C5A2F4F4F726353677A656335383 94733572B786A5642766B536F4878316D4C746843336A613745694556372F74 412F34766D646F6243394B454D594944547A4343413 0734341514577675945577655 4454C4D416B4741315545 42684D43526B6B78497A416842674E5642416F4D476 C5A 685A584E3062334A6C61326C7A644756796157746C633274316379425552564E554D52677746675944 5651514C444139555A584E3061585A68636D316C626E526C5A5

	851784B7A417042674E5642414D4D496C5A535379425552564E5549454E4249475A76636942555A584E3049464231636E42766332567A49433067527A51434241594D7 87051774451594A59495A49415755444241494242514367676745654D4267474353714753496233445145 4A417A454C42676B71686B6947397730424277457 74C77594A4 B6F5A496876634E41516B454D53494549482B447357562F38667854755333426755696831 6C33384C5573666F395A334B45726430674153625A42704D49485142677371 686B69473977304243524143 4C7A47427744434276544342756A4342747A414C42676C67686B67425A514D4541674545495062696D72704B3679717830356E4371512B6 E665174786B65624138554D626B4841597761794E586239544D4947464D48326B657A42354D5173774351594456515147457 74A475354456A4D4345474131 5545436777 61566D466C63335276636D567261584E305A584A706132567A6133567A494652465531517847444157426 74E564241734D4431526C63335270646D46796257567564475 66C644445724D436B474131554541777769566C4A4C4946524655315167513045675A6D39794946526C6333516755485679634739 7A5A584D674C5342484E414945426 77A476C44414E42676B71686B69473977304241514546414153434159437 34A316A5A3947516745786676 4C5831435A586151437574374763 4F33716C4375796A514367 65696B34317A537A754577333954357A476F6945354D46356E426F357962555351566A6B704654586B424730647178755A5755552F6D772F69477947346C75706B39517 549684A694341784D79513267367A7638445243533379774547467563377035 75746E466E7A3079636C5934424D426B704F62644552754362617 4514C644D436C7A3267 45537037515A747336505058627277494362613550395439726530 2B78554536447835306173 4D3254424D6F4152693559615673526674444 94D7341486F386445666C77 714870347065596E79386D79324C48685844776963635869746F2F534C2B537431454961787879503764783 8356E6C3145726E43386C6C7950367248645976 6B4A75555 5325654474E326A626863484F6A495255774556725845673755793962 4B752F654963466B36666E5A7133425A6B51316F6166494C5644484B7934755256382B48544C46 6B67364A79754F7078495178396751695834503677796B364F4C337674427252533 67A4A355734576E7569786270384E4B70687572757374696155416642 73534D45325 05A306D393651533572656E4235586F725669522F6C4D41424A5A35584C56717645432B6C706776706946755655427A6451452F3356614677513965624C34464A71357 8553D222C20227374617475 73223A20226F6B222C2022726561736F6E436F6465223A2022323030222C2022726561736F6E54657874223A20225369676E617475726520 67656E65726174656422207D
Plaintext payload	{ "version": "1.2", "signatureAlgorithm": "SHA256withRSA", "signatureType": "cms-pades", "signature": "MIIKugYJKoZIhvcNAQcCoIIKqzCCCqcCAQExDzANBglghkgBZQMEAgEFADALBgkqhkiG9w0BBwGgggcvMIIHKzCCBROgAwIBAgIEBgzGlDANBgkqhkiG9w0BAQ0FADB5 MQswCQYDVQQGEwJGSTEjMCEGA1UECgwaVmFlc3RvcmVraXN0ZXJpa2Vza3VzIFRFU1QxGDAWBgNVBAsMD1Rlc3RpdmFybWVudGVldDErMCkGA1UEAwwiVlJLIFR FU1QgQ0EgZm9yIFRlc3QgUHVycG9zZXMgLSBHNDAeFw0xOTA1MjAxMTI4MjNaFw0yNDAyMjUyMTU5NTlaMH0xCzAJBgNVBAYTAkZJMQ4wDAYDVQQKDAVUdWxsaT ESMBAGA1UEBRMJOTk5MDQ5NzEwMQ4wDAYDVQQqDAVERU5JUzEUMBIGA1UEBAwLSlVVVElMQUlORU4xJDAiBgNVBAMMG0pVVVRJTEFJTkVOIERFTklTIDk5OT A0OTcxMDCCAaIwDQYJKoZIhvcNAQEBBQADggGPADCCAYoCggGBAMhYM0U3szUviDon/lLqDugHLosOccs8hhbF8wiKRRz/LHpVY4Ys3vIum2cc4fz1hLg30hkZD50Xxicq H92WCVkmwV6nXei6nJzhNIu7AyX+fnZxgHCRpdykcxuUgcxBci4lXnDppkdUbJvbvF5RaXklJxpnkgCUnMK2aSQXCmCh0/nSfDDq3ocBEBhpJ8zQ+rSCGdqGbupegGwEN3c zHXbLNrO9qf1Ul1dgAsj4Ck0PK9Oicj6nxgGLbfj9f8yDnEVfIaGaPiaTB95+1FAleTsMT8YSYAc27CwFcHhdd01pLqb8jsDh9MAs1h92qtmLGCe2XwTk4qn194acDCwngeeOywk RbAxlGgv7mXml13i/EHjjz3SoYIToPfeV7IYnFIyu5ztv0Cx9gSriCpowM/A+R5AI0CVUPxARM3FPcvn09pw2vwEcUkN6ujdBPo4Z5y+HQbkVxw1rCSywcWGqUqP0QhTVMd4Oi rcZEhvQdy3ONfWEGycqT2H8jf7Zpw5V1QIDAQABo4ICNTCCAjEwHwYDVR0jBBgwFoAUPZqjtfgVEe8RyuvHXE2TgLLHP8EwHQYDVR0OBBYEFLFkwxLc3eyEe8b3Q16ps YLbpT5eMA4GA1UdDwEB/wQEAwIGQDCBzgYDVR0gBIHGMIHDMIHABgoqgXaEBWMKgUsBMIGxMCcGCCsGAQUFBwIBFhtodHRwOi8vd3d3LmZpbmVpZC5maS9jcHM 5OS8wgYUGCCsGAQUFBwICMHkad1Zhcm1lbm5lcG9saXRpaWtrYSBvbiBzYWF0YXZpbGxhIC0gQ2VydGlmaWthdCBwb2xpY3kgZmlubnMgLSBDZXJ0aWZpY2F0ZSBwb 2xpY3kgaXMgYXZhaWxhYmxlIGh0dHA6Ly93d3cuZmluZWlkLmZpL2Nwczk5MCUGA1UdEQQeMByBGmRlbmlzLmp1dXRpbGFpbmVuQHR1bGxpLmZpMA8GA1UdEwEB/ wQFMAMBAQAwNwYDVR0fBDAwLjAsoCqgKIYmaHR0cDovL3Byb3h5LmZpbmVpZC5maS9jcmwvdnJrdHA0Yy5jcmwwbgYIKwYBBQUHAQEEYjBgMDAGCCsGAQUFBzA ChiRodHRwOi8vcHJveHkuZmluZWlkLmZpL2NhL3Zya3RwNC5jcnQwLAYIKwYBBQUHMAGGIGh0dHA6Ly9vY3NwdGVzdC5maW5laWQuZmkvdnJrdHA0MC0GCCsGAQU FBwEDBCEwHzAIBgYEAI5GAQEwEwYGBACORgEGMAkGBwQAjkYBBgEwDQYJKoZIhvcNAQENBQADggIBAD7+WAav3WfDY2N3WIiXtgpSwmE/aHYSHBskU++mK5Uk 3B1dE4WGa+ku1hq3WvKNQ6PttGq7WcGWpdU4SV8hpYCQKcTJevYYoc9WkN4ZD6EncFJ/zS00/cOZRQlwIxdo5EmyYicWyANfKj19PLlVb6ELqKRG1vlElipCJrRgFPcJj+Cv WsxZ1d8N02nKajmGpG443x7J7oLxXNWOgBS6RsU262Cdn9OKkyKpg5iXip4PygyPIJ6cco0pYwD+2zqpbmuwDG2QLGGi+Y5T51C2vLvglFlKKq+2KBm9fRX/ehGN8Shj7eo v1MePAE7vntMCa4FY2h9h7U26+Z6cRQiuYllM+xqinxz0/LikdRpu4GDr62bQGxP5B42OUySlyzMeQLseoL7NHOtr4KRVaFD4qp65PIyq52SQotZlTHRqSShFPoEn4bfWP9oh4 NGRk0uKcFLrItEpixuQr+8eYDpmk1dMI0rs42QelT0gBx2A9/aUG6vVqqZzpONzZLOJ9JEHImASu+R/PSPtpL1KsaywZupqJWxGYNvtivXrSzNK5R/ikyZNkZ0iDRtqbFkkD8jpO mI5a2xdczI0f707rxJLiX8TD5Cc/7jv96+zqSjsmj2zsQkitflZ/OOrcSgzec589G3W+xjVBvkSoHx1mLthC3ja7EiEV7/tA/4vmdobC9KEMYIDTzCCA0sCAQEwgYEweTELMAkGA1U EBhMCRkkxIzAhBgNVBAoMGlZhZXN0b3Jla2lzdGVyaWtlc2t1cyBURVNUMRgwFgYDVQQLDA9UZXN0aXZhcm1lbnRlZXQxKzApBgNVBAMMIlZSSyBURVNUIENBIGZvciB UZXN0IFB1cnBvc2VzIC0gRzQCBAYMxpQwDQYJYIZIAWUDBAIBBQCgggEeMBgGCSqGSIb3DQEJAzELBgkqhkiG9w0BBwEwLwYJKoZIhvcNAQkEMSIEIH+DsWV/8fxTu S3BgUih1l38LUsfo9Z3KErd0gASbZBpMIHQBgsqhkiG9w0BCRACLzGBwDCBvTCBujCBtzALBglghkgBZQMEAgEEIPbimrpK6yqx05nCqQ+nfQtxkebA8UMbkHAYwayNXb9T MIGFMH2kezB5MQswCQYDVQQGEwJGSTEjMCEGA1UECgwaVmFlc3RvcmVraXN0ZXJpa2Vza3VzIFRFU1QxGDAWBgNVBAsMD1Rlc3RpdmFybWVudGVldDErMCkGA 1UEAwwiVlJLIFRFU1QgQ0EgZm9yIFRlc3QgUHVycG9zZXMgLSBHNAIEBgzGlDANBgkqhkiG9w0BAQEFAASCAYCsJ1jZ9GQgExfvLX1CZXaQCut7GcO3qlCuyjQCgeik41z SzuEw39T5zGoiE5MF5nBo5ybUSQVjkpFTXkBG0dqxuZWUU/mw/iGyG4lupk9QuIhJiCAxMyQ2g6zv8DRCS3ywEGFuc7p5utnFnz0yclY4BMBkpObdERuCbatQLdMClz2gES p7QZts6PPXbrwICba5P9T9re0+xUE6Dx50asM2TBMoARi5YaVsRftDIMsAHo8dEflwqHp4peYny8my2LHhXDwiccXito/SL+St1EIaxxyP7dx85nl1ErnC8llyP6rHdYvkJuUU2VTG N2jbhcHOjIRUwEVrXEg7Uy9bKu/eIcFk6fnZq3BZkQ1oafILVDHKy4uRV8+HTLFkg6JyuOpxIQx9gQiX4P6wyk6OL3vtBrRS6zJ5W4Wnuixbp8NKphurustiaUAfBsSME2PZ0m96 QS5renB5XorViR/lMABJZ5XLVqvEC+lpgvpiFuVUBzdQE/3VaFwQ9ebL4FJq5xU=", "status": "ok", "reasonCode": "200", "reasonText": "Signature generated" }

2.8 Reason codes

The reason codes for signature creation response messages are modeled after the HTTP status codes:

- 2xx series indicates a successful operation.
- 4xx series indicates that the operation was cancelled due to some reason, and
- 5xx series indicates that the SCS is malfunctioning or not supporting requested functionality.

The reason codes define general categorization of the operation result (e.g., ok, user cancelled, bad request), and are not intended to give detailed description to the HTML5 application. More detailed error handling should be done internally by the SCS.

The reason code related response parameters:

- **status** (string, mandatory): "ok" if the operation was successful, "failed" if the operation was unsuccessful. If the reasonCode value is 200, then status is "ok". With any other reasonCode value, the status is "failed".
- **reasonCode** (integer, mandatory): possible reason code values are listed on the table below. This field intended for general categorization of the error class, and should be used generate end user understandable error message in desired language.
- **reasonText** (string, mandatory): the textual error description. It is recommended that description takes the form "<Text>: <Description>" where <Text> is the text in the Text column in the table below, and the <Description> contains more detailed description of the error. This field is mainly intended for developers, and should not be shown to the end user.

Code	Text	Explanation
200	OK	Signature was created successfully
400	Bad request	Request was malformed, e.g., - unknown Request-URI - bad JSON format - missing mandatory parameters in signing request - a parameter contains a value that is not supported
401	Unauthorized	End user cancelled operation, e.g.: - end user did not select a certificate, - usage of the key was not authorized (e.g., PIN mistyped, PIN typing cancelled, or PIN locked) - end user does not have any applicable certificates
403	Forbidden	Request was denied, e.g.:

		- another request is being processed already - request was made from unidentified origin, i.e., https protocol was not used to download HTML5 app requesting the signature
413	Request Entity Too Large	Request size was too big
500	Internal Server Error	SCS is not working properly, e.g., is misconfigured
501	Not Implemented	Requested functionality is not implemented.

An example error response:

```
HTTP/1.1 200 OK
Content-Length: 88
Content-Type: application/json
Server: SCS 1.2 (2021-12-01)
Access-Control-Allow-Origin: *
Accept: application/json, */*
Date: Wed, 04 Jun 2017 07:38:16 GMT

{
  "status": "failed",
  "reasonCode": 401,
  "reasonText":"Not Authorized: end user cancelled operation"
}
```

3 Security considerations

This section is non-normative.

3.1 Authentication

The SCS can be used for end user authentication as well. A server creates a challenge request that has predetermined format, which is then digitally signed by the SCS using an end user certificate intended for authentication purposes. The authentication is validated by the server, it the signature is valid for the given challenge. The challenge should contain nonce and origin header contents, which the server should check to be valid to prevent replay attacks.

Additionally, the SCS could include a client nonce to the challenge so that the SCS would generate a different signature even if the challenge remains the same (similarly as described in section 3.7). This specification does not support this functionality.

3.2 Confidentiality, Integrity, Privacy

In this specification, the connection to the SCS can be protected, i.e., HTTPS protocol is used (cf. chapter 2.3).

Signature requests can be encrypted between server and SCS (cf. chapter 2.7)

3.3 Local server socket

The SCS server socket uses a server socket that is listening on the localhost interface. The implementers must make sure that connections to the SCS are allowed only from the localhost, i.e., connections from other devices should not be allowed. See also section 3.9 on Malware.

3.4 End user permission

The SCS should request for end user approval each time it receives a request for digital signature generation, i.e., it should not silently sign any content. Therefore the SCS will always show the certificate selection dialog when a signature creation request is received.

3.5 Identifying origin

The SCS should indicate to the end user the origin of the signature creation request. At its simplest terms, it can show the content of the HTTP header Origin, which is sent by the browser when it is making the CORS request to the SCS.

3.6 Non-repudiation signatures

In order to make non-repudiation signatures more meaningful, the end user should be able to approve the content that is being signed. In practice, the SCS should show the content that is being signed to the end user so that the end user can inspect what is being signed. However, for instance binary or xml content cannot be displayed to

the end user in understandable form. The current specification allows that content can be signed without showing the end user what is being signed.

3.7 Nonce

In order to prevent the SCS to sign data that is completely given by the requestor, the SCS could be able to add nonce data to the to-be-signed data before it is being signed. This is typically not possible as it would break that signing procedure. However, it is possible with XML Signatures where the <Signature> element can be appended with optional <Object> elements without breaking the application level logic. However, the current specification does not support this functionality.

3.8 Certificate types

Typically in PKI, the end entity certificates have three major use cases:

- authentication, where the certificate is used to authenticate the user (authentication is typically done by digitally signing a piece of data that has been constructed from nonce values generated by the parties involved)
- decryption, where the key is used to send encrypted data to the holder of the key (encryption is typically done by a shared key that has then been encrypted with recipient's public key)
- non-repudiation, where the certificate used to digitally sign a piece of data (this is analogous to handwritten signature, where the signer enters into an agreement with another party)

The current specification does not limit the use of the certificates based on their intended usage but only digital signatures are supported (authentication, non-repudiation) but the decryption (unwrap) functionality is not supported.

3.9 Malware

As the SCS exposes a server socket on localhost (see section 3.3), any local application, e.g., malware, is able to send signature creation requests to the SCS. As the end user is always prompted with a certificate selection dialog, these requests are not silently processed, but it will be annoying and essentially a denial of service attack as the end user will get these dialogs every time there is a request (unless the SCS is already processing a request). The current specification does not address this threat.

One solution to address this threat is that the SCS has an admin interface (UI), where there would be an authorization view. In the authorization view, there would be an authorization token shown (e.g., 20 alphanumeric values). The first HTML5 application that tries to use the SCS interface in a particular browser, the SCS would send a HTTP response 403 Authorization required. The HTML5 application would then have logic to ask the end user to provide the authorization token. The end user would go to the authorization view, copy the token, and paste it to the dialog the HTML5 application would be showing. The HTML5 application would then give the token to the SCS and the SCS would set a permanent cookie with the browser. Any subsequent request to the SCS by the browser (regardless of the origin of the HTML5

app) would contain this cookie, which SCS would always check. This means that the SCS needs to enable the use of credentials in CORS (e.g., use Access-Control-Allow-Credentials header). After the token has been used once in authorization, it is discarded, and the SCS generates a new one.

3.10 Multi-user environments

The current specification does not support cases, where the operating system is a multi-user environment. This is due to the fact that if the system has multiple users logged on, there also should be as many SCS instances running as well. As the SCS should be running in predefined port on the localhost interface, it is not possible to distinguish which SCS instance belongs to which logged on user with the current specification.

3.11 Browser policies

CORS access from a web page is typically limited to the same scheme that was used when the corresponding HTML5 application (i.e., web page) was downloaded. In other words, if a web page was downloaded using https, the subsequent CORS requests must also be done using https. Similarly, if the web page was downloaded using plain http, then CORS requests need to be done with http also.

Some browsers (at the writing of this document, Internet Explorer 10 and 11) may by default restrict the access to the localhost interface using CORS mechanism. In this case, the end user should enable access via browser settings or and a system administrator should enable access by changing browser policies remotely.

4 SCS Profiles

This chapter describes the SCS profiles, i.e., operation parameters for SCS instances. For instance, a SCS instance can be said to "implement" RSA signature profile in which case it must follow profiles specifications listed below.

SCS implementations are required to implement support for parameters that underlined.

4.1 Base profile

Abstract profile.

Server port: **53952** (https port). The SCS shall use the listed port for https based access.

HTTP methods: GET, POST

HTTP URI: /sign, /version

4.2 RSA signature profile

Extends Base profile.

Hash algorithms: "SHA256" (default), "SHA384", "SHA512"

Signature algorithms: "SHA256withRSA" (default), "SHA384withRSA", "SHA512withRSA", "SHA256withRSASSA-PSS", "SHA384withRSASSA-PSS", "SHA512withRSASSA-PSS"

Content types (to be signed data): "data", "digest" (digest allowed only for signing purposes)

Signature types: "signature", "cms", "cms-pades"

CMS signature content (required fields):

- version: 1
- digestAlgorithms: SHA-256, SHA-384, or SHA-512 object identifier
- contentInfo.contentType: CMS Data object identifier
- contentInfo.content: The signed data is included if the original request had contentType as "data". If the original request had contentType as "digest", then signed data is not included.
- certificates: Contains the signing certificate and certificate chain.
- crls: Not present

- signerInfo.version: 1
- signerInfo.issuerAndSerialNumber: The issuer and serial number of the identity certificate
- signerInfo.digestAlgorithm: The same digest algorithm object identifier as in digestAlgorithms field
- signerInfo.authenticatedAttributes (required fields, other fields may be included):
    - contentType: The same value as in contentInfo.contentType field
    - messageDigest: Calculated message digest of the signed data using digest algorithm specified by signerInfo.digestAlgorithm. This value is either calculated by the implementation (i.e, when to-be-signed data is given by the requestor) or the value was given by the requester directly.
    - signingTime: The time when this CMS object was signed
- signerInfo.digestEncryptionAlgorithm: PKCS#1 rsaEncryption object identifier
- signerInfo.encryptedDigest: The result of encrypting the message digest of the complete DER encoding of the attributes value contained in the signerInfo.authenticatedAttributes field with signer's private key.

4.3 ECDSA profile

Extends Base profile.

Hash algorithms: "SHA256" (default), "SHA384", "SHA512"

Signature algorithms: "SHA256withECDSA" (default), "SHA384withECDSA", "SHA512withECDSA"

Content types (to be signed data format): "data", "digest" (digest allowed only for signing purposes)

Signature types: "signature", "cms", "cms-pades"

CMS signature content

All fields are the same as in RSA signature profile except:

- signerInfo.digestEncryptionAlgorithm: ECDSA signature algorithm identifier (SHA256withECDSA, SHA384withECDSA, or SHA512withECDSA)
    - Please see

- signerInfo.encryptedDigest: the result of signing the message digest of the complete DER enconding of the attributes value contained in the signerInfo.authenticatedAttributes field with signer's private key.

4.4 CMS-pades profile

CMS-pades signature content follows CMS signature specification with two exceptions, defined in PAdES signature specification. Following fields are not present:

- signerInfo.authenticatedAttributes.signingTime
- signerInfo.digestEncryptionAlgorithm (=digestAlgorithm)