HTML5 and Digital Signatures

Signature Creation Service 1.3

1.3.2025

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

Document management

Owner	Jari Pirinen / DVV
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   1.3.2025

1.3	Security improvements for requests formed for authentication purposes: • Added mandatory semicolon to authentication request definition • Shared information has been added to the transaction-based signature to prevent a "man in the middle" attack. • Added new OIDs for CosmoX based smart cards	01.03.2025/SP

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

Table of contents

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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/
[RFC8446]	The Transport Layer Security (TLS) Protocol Version 1.3
[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

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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]:

  o Each HTTP response SHALL contain the `Access-Control-Allow-Origin` header with value "*".

  o 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.

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

- 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.

  o 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.

  o 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:

  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.

  Starting from SCS version 1.3, semicolon must exist between origin and nonce.

  Example challenge in string format:
  ```
  https://dvvtest.fineid.fi;7134bf29-ee63-4adb-83bf-
  7248eb90e6a13d247db9-2807-45c7-9395-
  f345d006a301066e3c40-8765-4c75-ba49-0aad82717c96
  ```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

Example challenge in base64 format:
```
aHR0cHM6Ly9kdnZ0ZXN0LmZpbmVpZC5maTs3MTM0YmYyOS1lZTYzL
TRhZGItODNiZi03MjQ4ZWI5MGU2YTEzZDI0N2RiOS0yODA3LTQ1Yz
ctOTM5NS1mMzQ1ZDAwNmEzMDEwNjZlM2M0MC04NzY1LTRjNzUtYmE
0OS0wYWFkODI3MTdjOTY=
```

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.

When using transaction based signatures, starting from SCS version 1.3 SCS SHALL append a semicolon and **serverKey** to the challenge_request before computing the signature.

> origin;nonce;serverKey

serverKey is in the same base64 encoded format as it was presented in the "Begin Transacton" payload section of the JWS message.

Example in string format:
```
https://dvvtest.fineid.fi;7134bf29-ee63-4adb-83bf-
7248eb90e6a13d247db9-2807-45c7-9395-f345d006a301066e3c40-
8765-4c75-ba49-
0aad82717c96;MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAERNUPyd8y
v8SJ98TPYpRqvDrOeq0E32Wa/jQsXgQzQivMv/gY+gKnFQVK4wtOUYYTM
Lyzc9ZaMXhhIpy3M19d6Q==
```

Example in base64 format:
```
aHR0cHM6Ly9kdnZ0ZXN0LmZpbmVpZC5maTs3MTM0YmYyOS1lZTYzLTRhZ
GItODNiZi03MjQ4ZWI5MGU2YTEzZDI0N2RiOS0yODA3LTQ1YzctOTM5NS
1mMzQ1ZDAwNmEzMDEwNjZlM2M0MC04NzY1LTRjNzUtYmE0OS0wYWFkODI
3MTdjOTY7TUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJemowREFRY0RRZ0FF
Uk5VUHlkOHl2OFNKOThUUFlwUnF2RHJPZXEwRTMyV2EvalFzWGdRelFpdm
k12L2dZK2dLbkZRVks0d3RPVVlZVE1MeXpjOVphTVhoaElweTNNMTlkNl
E9PQ==key
```

- 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:

  o Key algorithm type,

  o Issuer DN field,

  o KeyUsage extension values, and/or

  o 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.

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025
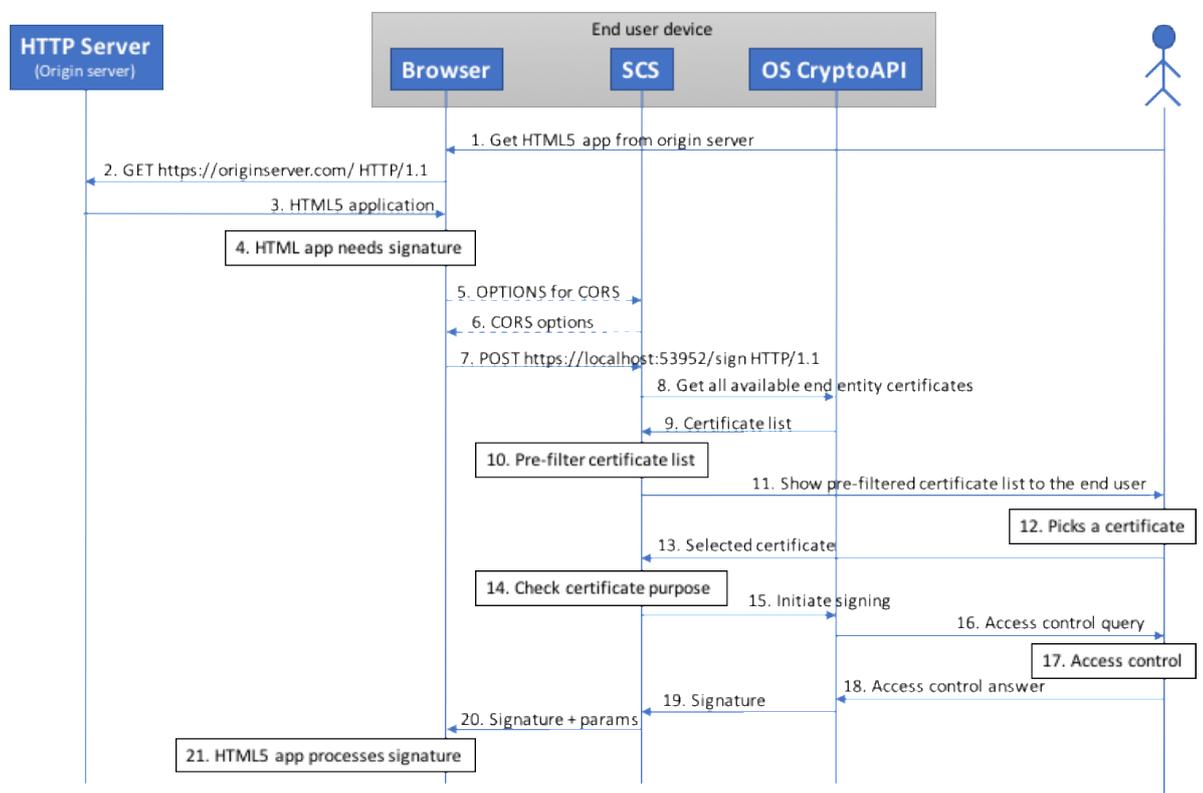
1. The end user types the URL of the origin server to the browser.

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:

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

- o The SubjectDN parameter SHALL include CommonName attribute containing "127.0.0.1".

- o The certificate SHALL contain SubjectAltName extension with at least following attributes:

    - extension is not critical,

    - dNSName: "localhost", and

    - iPAddress: "127.0.0.1".

- o The certificate SHALL contain KeyUsage extension with following attributes:

    - extension is critical,

    - digitalSignature key usage, and

    - keyEncipherment key usage.

- o 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
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

```
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 1.3 (2024-09-10)
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
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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 fillet 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 "1.3".

- **httpMethods** (string, mandatory): the allowed HTTP method for signature creation request. The mandatory value for this specification is "POST". Optionally, httpMethods may include GET method, resulting "GET, POST" response.

- **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. Only mandatory value for this specification is "signature". Recommended optional types are "signature-plain,cms,cms-pades". Other types may also be implemented.

  o "signature"

    ▪ 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 [X9.62]:

      • SEQUENCE { INTEGER = R, INTEGER = S }

  o "signature-plain" [RFC8446]

    ▪ 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.

  o "cms"; if given content is [CMS]

- ▪ "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.

  - o "cms-pades"; [PADES]

    - ▪ Designed to be used for PDF signatures.

    - ▪ 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".

  - o Example#1; if SCS contains only ECC keys, value is "ECDSA".

  - o Example#2; if SCS contains only RSA keys but doesn't support RSASSA-PSS padding, value is "RSA".

  - o Example#3; if SCS contains RSA and ECC keys and RSA key supports PSS-padding, value is "RSA,RSASSA-PSS,ECDSA".

  - o 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")

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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 (13.03.2023-->)
1.2.246.517.4.2.6	DVV FINEID CosmoX org. card
1.2.246.517.4.2.6.1	Cosmo X token for org. usage ("Toimikortti")
1.2.246.517.4.2.6.4	Cosmo X token for temp. usage ("Varakortti")
1.2.246.517.4.2.7	DVV FINEID CosmoX sote card
1.2.246.517.4.2.7.1	Cosmo X token for social welfare and healthcare ("Ammattikortti")
1.2.246.517.4.2.7.2	Cosmo X token for social welfare and healthcare ("Henkilostokortti")
1.2.246.517.4.2.7.3	Cosmo X token for social welfare and healthcare ("Toimijakortti")
1.2.246.517.4.2.7.4	Cosmo X token for social welfare and healthcare ("Varakortti")

Example HTTP response:

```
HTTP/1.1 200 OK
Content-Length: xx
Content-Type: application/json
Server: SCS 1.3 (2024-09-10)
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 "1.3").

- **selector** (object, optional) contains the certificate selector parameters

  o **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.

  o **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.

  o **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

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

insensitive. If specified, the end user certificate MUST have all the listed key usages present in the KeyUsage extension.

- o **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.

- o **strictKeyPolicy** (Boolean, optional). This selector is used 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. Default value is false.

- **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".

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

  - o 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.
  - o "RSA" stands for RSASSA-PKCS1-v1_5 signatures [PKCS1]
  - o "RSASSA-PSS" stands for PKCS#1 v2.1 padding scheme that is suggested to be used with new implementations
  - o "ECDSA" for EC keys [ECDSA].

- **signatureType** (string, optional) specifies the requested signature format. Possible values are "signature","signature-plain","cms" and "cms-pades", as defined in "**signatureTypes**"

  - o "signature" default value. Indicating plain raw signature.

  - o "signature-plain" indicates that the returned ECDSA signature is returned without ASN.1 encoding [RFC8446].

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

- o "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.

- o "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"
}
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

2.6.2   Request (GET, optional)

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 "1.3").

- **status** (string, mandatory) indicates whether the signature creation operation was successful.

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

- o "ok" indicates that the operation was successful,

- o "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:

  - o "SHA256withRSA"

  - o "SHA384withRSA"

  - o "SHA512withRSA"

  - o "SHA256withRSASSA-PSS"

  - o "SHA384withRSASSA-PSS"

  - o "SHA512withRSASSA-PSS"

  - o "SHA256withECDSA"

  - o "SHA384withECDSA"

  - o "SHA512withECDSA"

  - o 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 1.3 (2024-09-10)
Access-Control-Allow-Origin: *
Accept: application/json, */*
Date: Wed, 04 Jun 2017 07:38:16 GMT
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

```
{
  "version": "1.3",
  "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"
}
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

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.

Please note that version 1.3 contains two security enhancements that are related user authentication. Look chapter 2.1 for more details.

1) Server side; added semicolon between origin and nonce.

2) Client side; append serverKey to challenge_request before computing the signature.
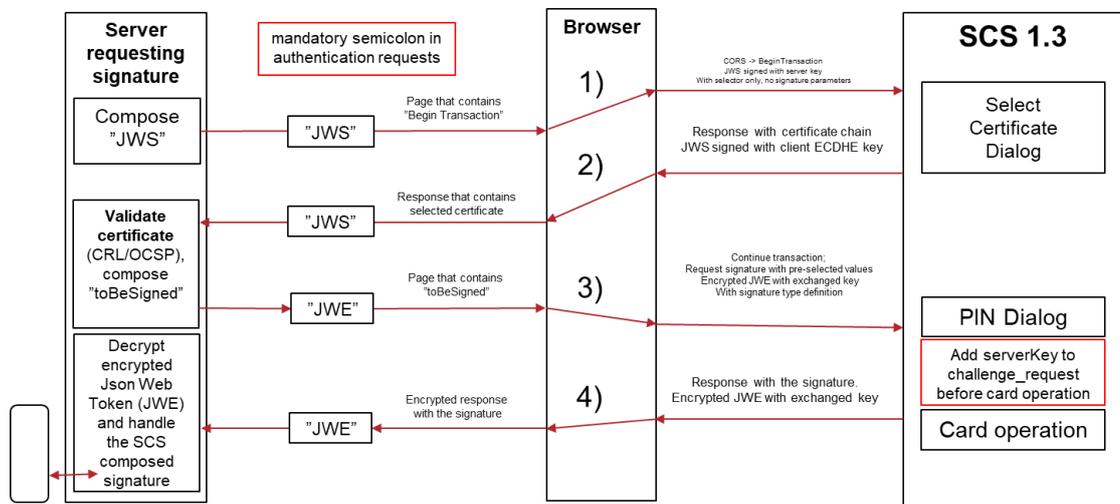
2.7.1   Begin transaction

"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)

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

2) server composes payload JSON message that contains following claims; *version, serverKey, serverCert, strictKeyPolicy* and *selector.*

- *version*, *strictKeyPolicy* and *selector* are encoded as defined in the chapter 2.6.1.

- *serverKey* string object (generated ephemeral EC keypair) should be encoded as follows;

  o 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].

    ▪ Supported *ECDHE* curve types are *secp256r1, secp384r1 and secp521r1.*

- *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
idmVyc2lvbiI6ICIxLjIiLCAic2VsZWN0b3IiOiB7ICJrZXlhbGdvcml0aG1zIjogWyAicnNhIiwiZWMiIF0sICJrZXXl1c2FnZXMiOiBbICJub25yZX
B1ZGlhdGlvbiIgXSB9LCAic2VydmVyS2V5IjogIk1Ga3dEd1lIS29aSXpqMENBUV1JS29SXpqMERBUWNEUWdBRVJOVVB5ZDh5djhTSjk4VFBZcFJxd
kRyT2VxMEUzMldhL2pRc1hnUXpRaXZZNdi9nWStnS25GUVZLNHd0T1VZWVRNTHl6YzlaYU1YaGhJcHkzTTE5ZDZRPT0iLCAic2VydmVyQ2VydCI6ICJN
SUlFQ0RDQ0F2Q2dBd01CQWdJSW1wY3AwcXhhVZS9vd0RRWUpLb1pJaHZjTkFRRUxCUUF3VXpFTE1Ba0dBMVVFQmhNQ1Jra3hHekFaQmdOVkJBb1RFa1o
xYW1sMGMzVWdSbWx1YkdEdVpQlBlVEVuTUNVR0ExVUVBeEllUm5WcWFYUnpkkU0JFYVdkcFUybG5iaUJNYjJOaGdJDQkRaWEowYVdacFkyRjBaVENDVU
VpDQlBlVEVyTUNrR0ExVUVBeE1pUm5WcWFYUnpkkU0JFYVdkcFUybG5iaUJNYjJOaGdJDQkRaWEowYVdacFkyRjBaVENDQVNJd0RRWUpLb1pJaHZjTkFR
RUJCUUFEZ2dFUEFEQ0NBUW9DZ2dFQkFMEEpmNnVEdXljZGIxZG1xR11YWl1RZ1BZamVySGVRdXZDem5UMml1IdFFPVDllaWxISG5RT3haOUY2Vzl3L24xRFV
IbW1RNjZoeEV1TExhHYXBzUXVUVUMxQXN0RHBmK0U4dDE0MzJDbHIwZjZPd2tJaklqZTJBYXYrR2o2VXRGZnVieU1nQ1VOTi9oV2FZTW11WitpNmFuY0
N0cUtBa3FempDRDdhYXJVOTNESnpFYzBMTHdEWktHY21MN2IyacVd2WTU0OVB5S0EwT0JicHRoeU4ySzJjMURQT0tWNlUzcThURjFQMWVyY3M3MSszv
jJLdytGdUQyRDNXdzI3TVl2ckx1MnF1SVJiK0dFZFB0QVQ4aGNaSGVJdmd6Z1cvU1ZxV1dVOUFvWDVIS2gzU3AxdXRiZDJxa2Q2Q2WjZPRUZMV01uWmlK
UzhiQTVIMEREpelpvvandaUkV6b2NDQXdFQUFhT0IyekNCMkRBZEJnTlZIU1VFRmpBVUJnZ3JCZ0VGQlFjEFjUlUld1lCQlFVSEF3SXdEZm1lEV1IwUEF
RSC9CQVFEQWdXdZ01DVUdBMVVkRVFFRZU1CeUNNDV3h2WTJGc2FHOXpkSU1KTVRJM0xqQXVNQz4aHdSL0FBQUJNRUFHQ0NzR0FRVUZCd0VVQkRRd01qQX
dCZ2dyQmdFRkJRY3dBb1lyaHYUhSMGNEb3ZMMnh2WTJGc2FHOXpkRG8xTXprMU1TOXpjMnhwYzNOMVpqYNVpZm0owTUI4R0ExVWRjd1FZTUJhQUZDdU
kzSEw5WDlpVzdtZXgvNDlrR1hOFbTRNQjBHT1FVZERnNVdCQlR3cE5tOE1CZlZ4OWdJdWektVtb21ldU1XRU5IWHpBTkJna3Foa21HOXcwQkFRc0ZBQU9
DQVFFQU11VDRpeXowcCUd4Ymw0Y09DME4xR0JwaEEhqQVBXZDcxMzVPTGdnbBNS3EzNElhOVzpeGdU5OHZlNDFCTUTU53SktMZ11wc3QzaVdvMVRFV11
tUnNiYVA2SHA2SHA2TWNpanhQdGdJXMjZSOHdxbFhnUnpxNCtvQVdtczhs RHpIbmp0L1kk2ako3MWJhcld5aHVMRVNsOTFuRmh1Y0M4aVNPazF4S0JleWxwZmlGMW
lPcXplQ2luc1JNOEs3ZzNiQkVRWUloWVZNeFB2UVQgybm9EdC82YU5ucjJHTGNycER3MDFoc3Q5amRlVGhGRzZBaHVBMWhpcXFNRkkyajJxN1Z6dUM1M
0FFSTdPVWRuU0RrTmFOeEhBK1VlWFBaTDlGZmRtQUN0aGt0Q1JWb0Y0bmR1Y21DOS9nUTRRRelJpdnptta1pVcDc0bEFiSVlSY2g5aUg4ZzNoeSSswQ2ta
dz09IiB9.SDckvDq-hGOeU4NvEUtDkK45Y2K-
z1PklbBh47pCMWJJan_e6Z8Tpleu_z9gPAjmuTZ32w3EZbAlOx4PLNKOG9332u24LvZ56ExtLOrvsEKh13yDAMxMCz4XhnuoRIFULb3oM1ytbrsLvoZ
HD042ZL_RHzFjhsfi2oIJ42j-Pf6pI91GFXBQwLB5ypSMEVL-
ob45M25DFYoSunlcCTVgqb5nwncltHxEBDKt4R7C6abfxY96GqJWekInnIZ4NWEONQLZibRgB6HKTNW9YNaGbOF5fCH37oHUyYzK1o97eNuXSR-
5lM2WqZVvvNPfV1g42ZDbrPxnmUUHUcrxbZNF4w

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

JWS Header:

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

JWS Payload:

```
{
  "version": "1.3",
  "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.

    o 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
idmVyc2lvbiI6ICIxLjIiLCAidHJhbnNhY3Rpb24iOiAiTUZrd0V3WUhLb1pJemowQ0FRWU1Lb1pJemowREFRY0RRZ0FFc04weFMrZjNrSFBxay8rRk
NJRlFPSnNSrjYzQUN1cy81R1UrbENjZ2Z4UlJ3SU5xc2JTWncxTGI3U1pIZFJxWGcyMWFYd3NjeW5ZajZOd2IwMm1pOHc9PSIsICJjaGGFpbiI6IFsgI
klJSUhLekNDQlJPZ0F3SUJBZ0lFQmd6R2R2xEQU5CZ2txaGtpRz13MEJBUUJGQURCNU1Rc3dEUVlEVlFRR0V3SkdUVEVqTUNFR0ExVUVDZ3dhVm1GbGMz
UnZjbVZyeVVhOMFpySnBhMlZ6YTNWeklGUkVMVMVF4R0RBV0JnTlZCQXNNRDFSbGGMzUnBkbUZ5YldWdRRHVmxkREVyTUNrR0ExVUVBd3dpVmxKKTElGUkZ
VMVFnUTBFZ1pttOXlJRlJsYzNRZ1VIVnlnJRzl6WlhNZ0xTQkhOREFFlRncweE9UQTFNakF4TVRJNE1qTmFGGdzB5TkRBeU1qVXlNSVFU1TlRsYSU1IMHhDek
FKQmdOVkJBWVRBa1pKZVE0d0RBWURWUVFLREFWWRXeHNhVEVTTUJBR0ExVUVCUk1KT1RrNU1EUTVOekV3TVE0d0RBWURWUVFxREFWRVJVNUpVekVVVT
UJJR0ExVUVCQXdMU2xWlZFbE1RVWxPUlU0eEpEQWlCZ05WQkFNTUcwZFWV1JKVEVVGSlRrVk9JRVVJGVGtsVElEVEazVPVEEwT1RjE1EQ0NBYU13RFFZ
SktvWklodmNOQVFFQkJRQURnZ0dQQURDQ0FZb0NnZ0dBUU1oWU0wVTNzelV2aURvXWi9sTHFEdWdTG9zT2NjczhoaGGdlpS1JSei9MSHBWWTRZczN
2SXVtMmNjNGZ6MWhMMzzMwaGtaRDUwWHhpY3FIOTJXQ1ZrbXdWNm5YZWk2bkp6aE5JdTdBeVgrZm5aeGdIQ1JwWjZHlrY3h1VWdjeEJjaTRsWG5EEcHBrZF
ViSnZidkY1UmFYa2xKeHBua2dDVW5NSzJhU1h1FYQ21DaDAvblNmRERxM29jQkVCaHBKKOHpRK3JTQdkcUdidXBlZ0d3RU4zY3pIWWdJMTnJPOXFmMVVsM
WRnQXNqNENrRMFBLOU9pY2o2bnhnR0xiZmo5Zjh5RG5FVmxZJmYUdGlhVEnI5NSSxRkFsZVRzTVQ4WVNZQWMyN0N3RmNIaGGrRkMDFwTHFiOGpzRGg5TUFz
MWg5MnF0bUxHQ2UyWHdUazRxbjE5NGFjREN3bmdlZU95d2tSYkF4bEndjdtWG1sMTNpNL0VIamp6M1NvWU1Ub1BmZVY3S1luRkl5dGV6dHYWQ3g5Z1N
yaUNWb3dn6dNL0ErUjVBSTBDlVQeEFSTTNGGUGN2bjA5cHcydndFY1VrTjZ1amRCUG80WjV5K0hRYmtWeHcxckNTeXdjjV0dxVXFQMFFoVFZNZDRPaXJJWk
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

VodlFkeTNPTmZXRUd5Y3FUMkg4amY3WnB3NVYxUUlEQVFBQm80SUNOVENDQWpFd0h3WURWUjBqQkJnd0ZvQVVQWnFqdGZnVkVlOFJ5dXZIWEUyVGdMT
EhQOEV3SFFZRFZSME9CQllFRkxGa3d4TGMzZX1FZThiM1ExNnBzMVoxIFZThiM1ExNnBzMVoxRHRdFQi93UUVBd2l0HUURDQnpnWURWUjBnQklIR01JSERN
SUhBQmdvcWdYYUVCV01LZ1VzQk1JR3hNQ2NHQ0NzR0FRVUZCd0lCRmh0b2RIUndPaTh2ZDNkM0xtWnBibVZwWkM1bWFTOWpjSE01T1M4d2dZVUdDQ3N
HQVFVRkJ3SUNNSGthZDFaaGNtMWxibTVsY0c5c2FYUnBhV3RyWVNCdmJpQpnZV0YwWVhacGJHEhGhJQzBnUTJWeWRHbGlhV3RoZENCd2IyeHBZM2tnWm
1sdWJuTWdMU0JEW1hKMGFXWnBZMkYwWlNCd2IyeHBZM2tnYVhNZ11lYWmhhV3hoWW14bE1HaDBkSEE2THk5M2QzY3VabWx1Wldsa0xtWnBMk53Y3prN
U1DVUdBMVVkRVFRZU1CeUJHbVJsYm1sekxtcDFkWFJwYkdGcGJtVnRSFIxYkd4cExtWnBQThHQTFVZEV3RUIvd1FGTUFNQkFRQXdOd1lEV1IwZkJE
QXdMakFzb0NxZ0tjJWWlhSFIWY0RvdkwzQnliM2g1TG1acGJtVnBaQzVtVYM5amNtd3ZkbkpyZEhBMF15NWpjbXd3YmdZSUt3WUJCUVVIQVFFRVlqQmd
NREFHQ0NzR0FRVUZCekFDaGlSb2RIUndPaTh2Y0hKdmVIa3VabWx1Wldsa0xtWnBMMk5oTDNaeWEzUndQzVqY25Rd0xBWU1Ldl1CQlFVSE1BR0dJR2
gwZEhBNkx5OXZZM053ZEdWemRDNW1hVzVsYVdRdVpta3ZkbkpyZEhBME1DMEdDQ3NHQVFVRkJ3RURCQ0V3SHpBSUJnWUVBSTVHQVFFd0V3WUdCQUNPU
mdFR01Ba0dCd1FBamtZQkJnRXdEUVlKS29aSWh2Y05BUVVOQlFBRGdnSUJBRDcrV0FhdjNXZkRZMk4zV0lpWHRncFN3bUVvYUhZU0hCc2tVKyttSzVV
azNCMWRFNFdHYStrdTFocTNXdktOUTZQdHRHcTdXY0dXcGRVNFNWOGhwWUNRS2NUSmV2WVlvYzlXaO40WkQ2RW5jRkovelMwMC9jT1pSUWx3SXhkbzV
FbXlZaWNXeUFOZktqMTlQTGxWYjZFTHFLUkcxdmxFSWlwQ0pyUmdGUGNKaitDdldzeFoxZDhOMDJuS2FqbUdwRzQ0M3g3SjdvTHhYTldPZ0JTN1JzVT
I2MkNkbjlPS2t5S3BnNW1YaXA0UHlneVBJSjZjY28wcFl3RCsyenFwYm11d0RHM1FMR0dpK1k1VDUxQzJ2THZnbEZsS0txKzJLQm05Z1JYL2VoR044U
2hqN2VvdjFNZVBBRTd2bnRNQ2E0R1kyaDloN1UyNitaNmNSUW11WUlsTSt4cWlueHowL0xpa2RScHU0R0RyNjJiUUd4UDVCNDJPVX1TbH16TWVRTHN1
b0w3TKhPdHI0S1JWYUZENHFwNjVQSXlxNTJTUW90WmxUSFJxU1NoR1BvRW40YmZXUD1vaDROR1JrMHVlY0ZMckl0RXBpeHVRcis4ZVlEcG1rMWRNSTB
yczQyUWVsVDBnQngyQTkvYVVHNnZWcXFaenBPTnpaTE9KOUpFSGxtQVN1K11vUFNQdHBMMUtzYX13WnVwcUpXeEdZTnZ0aXZYclN6Tks1Ui9pa3laTm
taMGlEUnRxYkZra0Q4anBPbUk1YTJ4ZGN6STBmNzA3cnhKTGlYOFRENUNjLzdqdjk2K3pxU2pzbWoyenNRa2l0ZmxaL09PcmNTZ3plYzU4OUczVyt4a
lZCdmtTb0h4MW1MdGhDM2phN0VpRVY3L3RBLzR2bWRvYkM5S0UiLCJNSUlIZXpDQ0JXT2dBd0lCQWdJREExdGpNQTBHQ1NxR1NJYjNEUUVCRFFVQU1J
R11NUXN3Q1FZRFZRUUdFd0pHU1RFak1DRUdBMVVFQ2d3YVZtRmxjM1J2Y21WcmFYTjBaWEpwYTJWemEzVnpJRlJGVTFReEtUQW5CZ05WQkFzUTlFTmx
jblJwWm1sallYUnBiMjRnUVhWMGFHOXlhWFI1SUZObGNuWnBZM1Z6TVJrd0Z3WURWUVFMREJCV11YSnRaVzV1WlhCaGJIWmxiSFYwTVI0d0hBWURWUV
FEREJWV1Vrc2dWRVZUVkNCU2IyOTBJRU5CSUMwZ1J6SXdIaGNOTVRnd01qQTVNVEF3TmpJMFdoY05Nemd4TURFd01UQXdOakkwV2pCNU1Rc3dDUVlEV
lFRR0V3SkdTVEVqTUNFR0ExVUVDZ3dhVm1GbGMzUnZjbVZyYVhOMFpYSnBhV3RyWVNCdmJpQnpZV0YwWVhacGJHEhGhJQzBnUTJWeWRH
dWRHVmxkREVyTUNrR0ExVUVBd3dpVmxKTElGUkZVMVFnUTBFZ1ptOXlJRlJsYzNZNRZlVIVnljRzl6WlhNZ0xTQkhORENDQWlJd0RRWUpLb1pJaHZjTkF
RRUJCUUFEZ2dJUEFEQ0NBZ29DZ2dKQkFNYUN0RXJBaStZYW5wM2pZeHBFdUtCd0JrZWYwek9LSGJ5czhtM3N2SWQ3aXU2eHBWb3ljUTBrR2w1TUxYSW
pSdUNOTWhRbkEvcFdZbjczLzRnd3pRMFpXdkwzbjNIaWptcUd1aDFraHNXSElDZDl5Q2pVR0FNT2MweXdsc25QNzhvcEhwakNDT3JqYkkzZGtVQkRub
VFvY1NGUG95SG4wQ11yRitjaDNlbUpWZGRuc29Way9FSkl5ejZ0NmFtcUkzVVVrMGJnY0VqakU2SEM1aDVVOHFHVEFsVzBLQzFRanRZQnRtV1c4MWVh
czZKMU4xaHp6M1JWWHZmc09ZbjF3eUFZQi9DYmE5Zkk2MEE2TXZGcWgvbUJXc1hXblgzL3JwdzgzWlMwTmFmWlcvVGV1Z0hhdStjaVRlcGxPRGE2eHp
zVUEreFhWeW4zWkwvcUN2WnBnVldjek42OFpOMzY5SGRDMSs3ZWE3UzIzYXkxb2t5MW9nbjlVaEVCNGljVnpFcjJjS09ZWEhITEFEbUhORllnRUkxa0
VoUmVBbFRNNGZKbm5kZ0dZRGMrRCs5aFJiK1R2dHMyWGRxdVVndnJtNVczbnNxOTRpbTRNYWdJd2pQZHZpc1hMSDM2MTk2Rk01WlIaWExzaTdQQkd5a
kczQ2dMQmpZWVZQaDZHKzJkSHBrd0x6bFJ4UTRnczBIaEk0L1B2TzNTb05vMzI5NE9aQUsyS3RxUzM2ZW1aakFEK1RJaWJtbFNhbXhYOXhHMkYyZTJX
Z11JNXk0NmcvN0NXa1dIR3pCMWJablM1aEhoM2tvMDZTalNMM1VQRS9tL2pYOEZvc2FXbE1ocDJIMGx2R1RmckxYeDhKbWxjWCtramlDdlF2MDcwb3F
yZmtDZUNxMVh6cno3ZEdvWEFnTUJBQUdqZ2dIcU1JSUI1akFmQmdOVkhTTUVHRFXZ0JTRlBhRGgvTEVKSitUYXZSK0dpMEFHY2tJS2dUQWRCZ05WSF
E0RUZnUVVQWnFqdGZnVkVlOFJ5dXZIWEUyVGdMTEhQOEV3RGdZRFZSMFBBUUgvQkFRREFnRUdNSUhPQmdOVkhTQUVnYll3Z2NNd2djjQUdDaXFCZG9RR
ll3cUJTUUV3Z2JFd2ZVUdDQ3NHQVFVRkZJ3SUNNSGthZDFaaGNtMWxibTVsY0c5c2FYUnBhV3RyWVNCdmJpQnpZV0YwWVhacGJHEhGhJQzBnUTJWeWRH
bGlhV3RoZENCd2IyeHBZM2tnWm1sdWJuTWdMU0JEW1hKMGFXWnBZMkYwWlNCd2IyeHBZM2tnYVhNZ11lYWmhhV3hoWW14bE1HaDBkSEE2THk5M2QzY3V
abWx1Wldsa0xtWnBMMk53Y3prNU1DY0dDQ3NHQVFVRkJ3SUUGaHRvZEhSd09pOHZkM2QzTG1acGJtVnBaQzVtYVM5amNITTVPUzh3RWdZRFZSMFRBUU
gvQkFnd0JnRUlvd0lCQURBNUJnTlZIQjhFTWpBd01DNmdMS0FxaGlob2RIUndPaTh2Y0hKdmVIa3VabWx1Wldsa0xtWnBMMk5ZYkM5MMmNtdDBaWE4wT
W1FdVkzSnNNSFFHQ0NzR0FRVUZCd0VCQkdnd1pqQXZCZ2dyQmdFRkJRY3dBBWVlqYUhSMGNEb3ZMMjljqYzNCMFpYTjBMbVpwYm1WcFpDNW1hUzkyY210
MFpYTjBNbU13TXdZSUt3WUJCUVVITUFLR0oyaDBkSEE2THk5d2NtOTRlUzVtYVc1bGFXUXVabWt2WTJFdmRuSnJkR1Z6ZERKakxtTnlkREFOQmdrcWh
raUc5dzBCQVEwRkFBT0NBZ0VBRkhpODVURnRF0V1I4bWJCREdhcExMUjRmRjE0dWJ5eWxBQ1JESHBKYXAyTGZTOHAxTEIyRmtoRGFzdnFITERpWUNaaE
huWkFyd0hpaEpCcmlVbGFPY2ZjaHd5WkxYSnNFdERScWRNdTBjTU9sYkt1QU9uZkFsdnU2SnFybXdtemhOTnNJNTJhME1obFA3TDVRd3oyU3VpWkJLR
U00NnB5VXIyU3JaTVAzZG00WWNXQi91UUpoMGtNbS95TWQwbEp0MM45R0owUUlKTUtlelNlYmNVb0MycFFyeUhZaElTcTVaSDI4ZFY5SkNyN1EvbnZN
eForY3cyaFJzY1RtdS9mYlk3bXIxNm5mQXNiaHM4TjhnQ2lEWjArUEpXZE9ydHowZDJvaGt6VnhZRlhFTTdRSjM2aEhKNS9ja005TGFsdi9iTzhUUmF
HM2hlRFJPT1llSTjFOdnFFRXYrazFmYmdEZVdJOHdMcD14ZmhOZVB5cTVjRmpWczFtRXV4VkxSSFEwOGxDQ21IcG14bUsxK0g2REJSTVRtMGI1VU9tMz
FpRnFMOXQ4VFJGeGhBelBuY1JQNXZwcjBYWDdZTEt1YUJ5VmNIR1FGemZMUHZLdEx6T3JZRy81Z01Kb1FaU3hyU0JmVVZwbis2Um13aytKMjF1bUdsN
2FzdHkxNFc5UzBRSVFJTjNEdHlhbTdXdGtGYVEwOW1uZFZ4bGNKTGVpZjRITWsrWWp5Unh4Ym44L3FtNjkva0ZyV1NZVkhGRE5SVHpOM2d1a2NSWCtp
ckFoRVZzRnJWUW1CQTRieHptZWVIOEc0Y0p3Y0dzS3ovZG93eEozQnVWMjFNcExIRzZZUjZtdElYVWJaeHpzamRIb3RvaHRaeG1KNm4yeTVuUXRHVzd
FN1dKanlSTnhRYjA9IiwiTUlJR0VqQ0NBL3FnQXdJBAgIE...

JWS Header

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

JWS Payload

```
{
  "version": "1.3",
  "transaction": "MFkwEwY...
  "chain": [
    "MIIHKzCCBROgAwIBAgIE...",
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

```
    "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 contains following claims; *version, content, contentType, hashAlgorithm, signatureAlgorithm, signatureType and displayText*.

- *version, content, contentType, hashAlgorithm, signatureAlgorithm and signatureType* are encoded as defined in the chapter 2.6.1

- *displayText* string object is UTF8 encoded and it is displayed to the user when requesting PIN code.

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-
```

Kehitys ja tietohallinto / Jari Pirinen

1.3.2025

```
voiDEpC_U7Zp9a_cSGolJJPdB57v587NI6wdvONdA7Ce2oG7v2HdRw.DpS-
ylmtrvDHI4b5WHsX9A
```

Example request details and test vector for developers:

JWE element	Value
Header	eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdTSIgfQ
Decoded header	{ "alg": "dir", "enc": "A256GCM" }
Encryption key	ECDH key exchange result: 658F205270C4C5C6C7217A039870C0337DEA1F41662DFF57AC3A8BA7DDFCBC8B
Initialization Vector	UdIVcZ_vwoW8Po72
Payload	K4Ob5TTmJdAxAgmph2YRtxAekVlowUOpaAqEjduRWFo1q_6bTozqgDUIzCw3ktR9dP_343uCp8FoopgOxlxpEo6DpBjHVp33CVSnb11EbLzGvaJGWJVMD_o94He0z3cvpzY8ZTkNi9Zk1RQFo-vWpmU-7wGb0ZM1z-MnvvY4AzmxiqMIRyu5urW3MGKWH8skBHJiaotJUEqi07T-voiDEpC_U7Zp9a_cSGolJJPdB57v587NI6wdvONdA7Ce2oG7v2HdRw
Authentication tag	DpS-ylmtrvDHI4b5WHsX9A
Decrypted payload	7B202276657273696F6E223A2022312E32222C2022636F6E74656E74223A202266344F785A582F782F464F354C6347425734B48575866777453782B6A316E636F5374335341424A746B476B3D222C2022646973706C617954657874223A2022547265616E73616374696F6E2074657374222C2022636F6E74656E74547970065223A20226469676573744222C202268617368416C676F726974686D223A20225341313536222C20227369676E617475726554797065223A2022636D732D706164657322207D

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

Plaintext payload	{ "version": "1.3", "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 1.3 (2024-09-10)
Access-Control-Allow-Origin: *
Accept: application/json, */*
```

```
eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTI1NkdDTSIgfQ..3-
ceXbhoZ6ReqFGF.3VgpdZZAdJrP7EXt691QAjTthPM8kxcGcvqxlmPQPopE5tf0hMPMLLKwut_CIH2iHZ17ke2oTg3_SiCyvfnfRgEuk83qCO2QYwn9
ZnX1dnazR02RuU_FplSXdMiRRk2cUWZb8NoJ_Y38QufO7TKCOlfEfgi1dF7KxC3EXUurEy1wZ4wDPma49wNHiwZfLXx1NRjUuja96VNwQ4yfC8hriHg
BluuKaRATd5AgGBW-7ApC6TPdm7xukzQfpiOvG3nRgUgqpeVcEwquSW51LfX-VQcR-9L2jwrtL88rTr1IXZgyBN7psQmbqiIuYqAZY2S-F2c7-
O7VCF_fmpHzxPm0QST0N7CV2_fZJLMh6U_O-
xeTyPd7HbBuTxZ8rtWhwTDdvarIMofrQAt2j__yKNgtIZT9LiDzYvL1u27tQskjY8ghK82NiwZBtGdOT9zkM2MBnYmTs8SFAOyxhjUSM5G553-
IARat-YBtXR_46AOd6uDbZ8nWbm-dtWIpab02yd_7CtEk6CSAEzQO7wEOa8fnNwdEKROgNR5EAMnoF9Qs-
iwnvyLcBm2621pQ8Mo71LMJA4Wbys9q8QZLWKqO-
a79qLO1kH_zV7QL_JBSsWJcj7DvhNbOz2crOmriiGcn5XPSs8VhmCm_kSlps7cLsGFBnwVKvQSWBL3nLao7Mn7Rr-EndLHRcgI3pHU_5sZj-
SaBqbpgbaHJjiYGV2fTMh9P_Zt4dfoTyIeZiv_2pxB-
EnUL4F6hyUCo0Tqz0efRqba0Ho3yXa1GwOUaC7CJx2sKhS8MBNCR_EalDoGzF4tUJNtMyRxMT6daiwdrFznxgb-
noqsob8w4mowZ_6gmGAIRLB2CrJH16QhOzt_R1Y0sv-Ad4agfikXWtKITWUC9wUxo6pNz1WUSE8Mte9S0SF13iJjsyFH9ms87KaYYU_GsHclKn8JPL-
GeKh-1On_n2ELBU1H3_n6PX6q1lbXa-vTqtw0y8ee3GBKWFUM0WJLKTH-
1qNpYS64UJGUHzbGqCNGa20YBLJCeF7O3DIEB6zYMxoYQIdbHaN5y0a4zffFyqXQE8zcgXP7AZ6G7QbPpmKw1JiixowaX7hWpx3b_0kDKKan9M7tZKx
qSZ3L8tVg_soHNzdDIr2sLYWVp07YKpglO2ooMCYlp3OsH6hOXfngme8Xt_cdSY00ZlZDcoHIHGdFOnUJ6HoSBgGI4nwYMA8d5lSvVYVb7_vVGkqhFX
bHtd15xAT6u-WuBYJ0DlsXyWdhPDUKZR0UEUISNyOn-ZGBA3aeuAG81NpRLaFk6xqeXuz5QQinRu9knvQaIymFHQ5p4GCexYg767LvNOUW2W-
_J5dKUy11knB36tyAQFz-g4qOEzvo2drX6nfU4Zv-Xvy_4GsVKRcBr1IRz4n9sTGd0Eix-IBTvTpL3-6vtw7UIdQrz-GS7pdFj-
R2PNW1Z758Sqa06oQikeU1bqNcDfN-VIDk09iomETDR1o4j003QaUy-VmIZ6OIG-
Ng208zUOgk6v_lfVIiCckStXhptWFoikstWo3fAcKcNOuFxIXE2QqHYhr8itEMTiiRe0eu5PxsyVOMoVaBwz-
qbuioD1lU7PpOvq6L995q4s71WssHWfR3jK9JMHVuTp6f5p2Ol7UW1Gil7MtHSsmjOx_F3DoRWnxQe8NqCuVSAo6NOYq5W323kXLpTc0FD143TwDPh_
AZT9zPXiB0_uKYvmwF0B05Ds28DnJyJryr06A41M3IO_ORL-fgCChvfmzHqKfTsu_i5-
ieVwrtXGMHwEmi4LtsQw83_xbPmejDeYwcjxH5vrtTYBn21kysfGU-KBM2MDWxys8ZZdTi99t6xVFC9k3rfdpDK2ryqjdysHzY1mUtso5rhDf-
fxMyGNC94i5T6Skh98F_7p-qtpLoik_bGDUqEH51n5HxHANogBISFR2nqdm_yAfmzjOVzqheNMJPN5BuyTCB6XNE9Hm0P0SFxu35MAFZROW-
OqHpyMWnaiJ4XNUiSMOVA0SdNmo73z0wQbcCucMyxmiHzMFW0S2ZackSBJDcbELnMPupi7rfnpB1AhjmHc_Wb-k8yFTyg8M46ND-
AvIuXY0coDo5jPJSuhaK1eBxbt0zyxxFOiNwWbEvVDFsMkfsf5EeMy9i1E61ZMeItf7oRNpJGMwqcXX-
Cz7jFLExIdZb8AJ8ry3cEW8trWzqFr6ixnIXWm_S5D2fiYr2QeFsTCvE816ukuErIee-S1_CaU81FX4r3d3JaxRDoXNG-
DOBB6kRvn2Q3DY2YUY2gBkTz7Kdco9a_B4gS9_M1tv3hD1XLBO0R_SOAEcB_qrQZWRoR5r8_krsLA8gPR5IaOqTd6cqB9Ayxu23z9-
PTLmDelpdsyuIOaEjkeuW9iA8MOZeAlp3kLcoehHjLc7qrdPho4fBhvbVWa-ckSZYyEd8S7YZwjGCdbFmjbcHIJ66GDNnBFdraDsba11ztdSLpv-
CkQJKCvRor2s3nYWqIYCcdE5C20hNW4AoXESUcYHCuMtJQINKLo686190T_ixuquTYgcYY6sCO2kQEwfsY_0uVPDzIWlIPa43VHeGa6VHOwodOjZ_tW
OWnnKqIFVHRzytYdhN3U4WtOCwj9QIi6biMi0dtgJRmXYgoSwz724-
bx2KXjveG9lsHG4QqB_6H11DKtsfsLOzd5tQYjOaWgltDq1aZu1VD3DXpPboVinVCkNp4G-FHyupyCnUMuZD-uKZePU0ipl-f0jbEZpY-
mFVnU8Y65mD-7b-ID-njydxymXMKLLtb8P_I5Ks12LC3YeeNW_1SUUY4aEtZc8IzxIGxLlNKmISfcW-
iSfgKJMuWiffcjZedmrrj49rsWOdY1C1xyjyNcY3_b4IQdGA4cydh8AZZDOifAfVVE8nWNoqmZDvZTWdfjbWVHZLmjDCEAnGz9UYB0Ews5m6ZiBIjSQ
YkfIVlyvOPLoeqHG-ZqPOr2JLek5MuXg9hnKg8r_woih4_bTstksIUWD7MBIupx1EGaGtEplCvObKHerxEnPTbLsmjz_CelBMJ-
qtaWG5l_gC_ftoDiFifRfFQbdrj5_BOO-ROcyiD33K-929gyrH3q5napyk-ibTAP-
kOWmEtTjsebnrFWSfSeKAN4GN0DgkFJ24GwYnORmxsZyTOdFBU2w8yO85hbVZXzEzX6EVtAVi30HCsn8ev1pCDWCCXpIlw8IxiWlUcuFNCAO8evdC4H
7LBx3rEIxzPOoESJrOSh-
L9lKwr4lYJaFihEC9uE5eZZvhePFVCoBRMoaqJK815W6fCJ7OhLVO0CXS3eaiUHthvHakZ9VWlg2_BwB06E2htPkyQ0H49WqQCzLbTVrYjPIuACaNGkM
WCG6UfoawzMVYsn_RmRrHIco0LVhAVr9uJOFhnPcOWt5dqeX7ujZ7gxFiFXLRxYRe3ClE-
nz8aHaXXbCiBduTbanurpMoUUjVh4C2rvduajK5vG7k_aKHmUKZKGK0eAlFdSY_tQqwdB4fIu9RixhNsJhsMBh_FDxR4jZMM0Kw6u89hv0OQKQLXSVX
x98Y_SDn1PUb9NiV2JNweHtAPfBAErFTZyAtn9t0SGA9Drx7Mx972a85NloTDECMQnjBL74jiGgs_Hii0_pOK1W79pqHIx8d1T83sXTBLeM_MhBfR53
zmLZLJ8PIym_M8NypgQQoezS0nJhtgzz_SJ_A3EEnRvSQ210hBSqWO7xntX N64ZxBbfqBJyCW9qcYWM5gR3h7ImTnpqz3nIX6POusBDp7QVLmdfSLKD
fdSguWwpAm6A7e7rCt__cwrAqPdUjtsCa9fE4z7t5ppv5UexA2MwvcGYdDls47uQ3O3NWbFFO2A-
RJTTQwn00y0A41soZXk4YNhZSUb5we9S_sNSi1N5z2G6g1J921JTucJ5hifADuPsBoQ2qBmhwYdvmhI2_6MqEQwQCoJa2cW-
nWIOra2dqMbUyTPv8C3sQgkckIpEOE0PjmkyL4WUztLe5MSFw7yBQcQ-BCNLYxvti6mKOsmA5iZ__JqFVKmK31G3nDMP8puHvyDFCXNEL3gLT12ju-
1Dyf2zvv7TPyZs8LKgiV2gtOgw38s_EfBXdiez-DWobTGAIMmkUYIjsWWQJwqkOQPLvfPL9KLmKjZlcCmEOO0mk6Nh4qRcUbHXPRkA-
Gdas4WFtfl3MQ3AaHyxswQCXTfzE0Yga5Our7qQf2w9vbw1woNAemk8SeZhDIyhgZlmjulQ9Iz5eeRq0aff2sedsehZBOBH6dtEx0DaGYQe9t9uu1pL
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" }

Kehitys ja tietohallinto / Jari Pirinen 1.3.2025

Encryption key	ECDH key exchange result: 658F205270C4C5C6C7217A039870C0337DEA1F41662DFF5 7AC3A8BA7DDFCBC8B
Initialization Vector	3-ceXbhoZ6ReqFGF
Payload	3VgpdZZAdJrP7EXt691QAjThhPM8kxcGcvqxlmPQPopE5tf0hMPMLLKwut_CIH2iHZ17ke2oTg3_SiCyvfnfRgEuk83qCO2QYwn9ZnX1dnazR02RuU_FplSXdMiRRk2cUWZb8No J_Y38QufO7TKCOIfEfgl1dF7KxC3EXUurEy1wZ4wDPma49wNHiwZfLXxlNRjUuja96VfwvQ4yfC8hrlHgBiuuKaRATd5AqGBW-7ApC6TPdm7xukzQfpiOvG3nRgUgqpeVcEwquSW5lLfX-VQcR-9L2jwrtL88rTrilXZgyBN7psQmbqijuYqAZY2S-F2c7-O7VCF_fmpHzxPm0QST0N7CV2_fZJLMh6U_O-xeTyPd7HbBuTxZ8rtWhwTDdvarlMofrQAt2j__yKNgtlZT9LiDzYvL1u27tQskjY8ghK82NiwZBtGdOT9zkM2MBnYmTs8SFAOyxhjUSM5G553-lARat-YBtXR_46AOd6uDbZ8nWbm-dtWlpab02yd_7CtEk6CSAEzQO7wEOa8fnNwdEKRQNgNR5EAMnoF9Qs-iwnvyLcBm262lpQ8Mo71LMJA4Wbys9q8QZLWKqO-a79qLO1kH_zV7QL_JBSsWJcj7DvhNbOz2crOmriiGcn5XPSs8VhmCm_kSlps7cLsGFBnwVKvQSWBL3nLao7Mn7Rr-EndLHRcgl3pHU_5sZj-SaBqbpgbaHJjiYGV2fTMh9P_Zt4dfoTyleZiv_2pxB-EnUL4F6hyUCo0Tqz0efRqba0Ho3yXa1GwOUaC7CJx2sKhS8MBNCR_EalDoGzF4tUJNtMyRxMT6daiwdrFznxgb-noqsob8w4mowZ_6gmGAIRLB2CrJH16QhOzt_R1Y0sv-Ad4agfikXWtKITWUC9wUxo6pNzlWUSE8Mte9S0SF13iJjsyFH9ms87KaYYU_GsHclKn8JPL-GeKh-1On_n2ELBUIH3_n6PX6q1lbXa-vTqtw0y8ee3GBKWFUM0WJLKTH-1qNpYS64U1XOJHtzhGqCNGa20YBLJCeF7O3DIEB6zYMxoYQldbHaN5y0a4zffFyqXQE8zcgXP7AZ6G7QbPpmKw1JiixowaX7hWpx3b_0kDKKan9M7tZKxqSZ3L8tVg_soHNzdDlr2sLYWVp07YKpgIO2ooMCYlp3OsH6hOXfngme8Xt_cdSY00ZlZDcoHIHGdFOnlJJ6HoSBgGl4nwYMA8d5ISvVYVb7_vVGkqhFXbHtd15xAT6u-WuBYJ0DlsXyWdhPDUKZR0UEUlSNyOn-ZGBA3aeuAG81NpRLaFk6xqeXuz5QQinRu9knvQalymFHQ5p4GCexYg767LvNOUW2W-_J5dKUy1lknB36tyAQFz-g4qOEzvo2drX6nfU4Zv-Xvy_4GsVKRcBrllRz4n9sTGd0Eix-lBTvTpL3-6vtw7UldQrz-GS7pdFj-R2PNWIZ758Sqa06oQikeU1bqNcDfN-VIDk09iomETDR1o4j003QaUy-VmlZ6OlG-Ng208zUOgk6v_lfVliCckStXhptWFoikstWo3fAcKcNOuFxIXE2QqHYhr8itEMTiiiR8oeu5PxsyVOMoVaBwz-qbuioD1lU7PpOvq6L995q4s7lWssHWfR3jK9JMHVuTp6f5p2Ol7UW1Gil7MtHSsmjOx_F3DoRWnxQe8NqCuVSAo6NOYq5W323kXLpTc0FD143TwDPh_AZT9zPXiB0_uKYvmwF0B05Ds28DnJyJryr06A4lM3IO_ORL-fgCChvfmzHqKfTsu_i5-ieVwrtXGMHwEmi4LtsQw83_xbPmejDeYwcjxH5vrtTYBn2lkysfGU-KBM2MDWxys8ZZdTi99t6xVFC9k3rfdpDK2ryqjdysHzY1mUtso5rhDf-fxMyGNC94i5T6Skh98F_7p-qtpLoik_bGDUqEH51n5HxHANogBISFR2nqdm_yAfmzjOVzqheNMJPN5BuyTCB6XNE9Hm0P0SFxu35MAFZROW-OqHpyMWnaiJ4XNUiSMOVA0SdNmo73z0wQbcCucMyxmiHzMFW0S2ZackS8JDcbELnMPupi7rlnpB1AhjmHc_Wb-k8yFTyg8M46ND-AvluXY0coDo5jPJSuhaK1eBdxbt0zyxxFOlNwWbEvVDFsMkfsf5EeMy9i1E6lZMeltf7oRNpJGMwqcXX-Cz7jFLExldZb8AJ8ry3cEW8trWzqFr6ixnlXWm_S5D2fiYr2QeFsTCvE816ukuErlee-S1_CaU81FX4r3d3JaxRDoXNG-DOBB6kRvn2Q3DY2YUY2gBkTz7Kdco9a_B4gS9_M1tv3hD1XLBO0R_SOAEcB_qrQZWRoR5r8_krsLA8gPR5laOqTd6cqB9Ayxu23z9-PTLmDelpdsyulOaEjkeuW9iA8MOZeAlp3kLcoehHjLc7qrdPho4fBhvbVWa-ckSZYyEd8S7YZwjGCdbFmjbcHUJ66GDNnBFdraDsba11ztdSLpv-CkQJKCvRor2s3nYWqIYCcdE5C20hNW4AoXESUcYHCuMtJQlNKLo886l90T_ixuquTYgcYY6sCO2kQEwfsY_0uVPDzlWllPa43VHeGa6VHOwodOjZ_tWOWnnKqIFVHRzytYdhN3U4WtOCwj9QIi6biMi0dtgJRmXYgoSwz724-bx2KXjveG9lsHG4QqB_6H11DKtsfsLOzd5tQYjOaWgltDq1aZu1VD3DXpPboVinVCkNp4G-FHyupyCnUMuZD-uKZePUOipl-f0jbEZpY-mFVnU8Y65mD-7b-lD-njydxymXMKLLtb8P_l5Ksl2LC3YeeNW_1SUUY4aEtZc8lzxlGxLlNKmISfcW-iSfgKJMuWiffcjZedmrrj49rsWOdY1C1xyjyNcY3_b4lQdGA4cydh8AZZDOifAfvVE8nWNoqmZDvZTWdfjbWVHZLmjDCEAnGz9UYB0Ews5m6ZiBljSQYkflVlyvOPLoeqHG-ZqPOr2JLek5MuXg9hnKg8r_woih4_hTstksIUWD7MBIupx1EGaGtEplCvObKHerxEnPTbLsmjz_CelBMJ-qtaWG5l_gC_ftoDiFifRfFQbdrj5_BOO-ROcyiD33K-929gyrH3q5napyk-ibTAP-kOWmEtTjsebnrFWSfSeKAN60DgkFJ24GwYnORmxsZyTOdFBU2w8yO85hbVZXzEzX6EVtAVi30HCsn8ev1pCDWCCXpllw8IxiWlUcuFNCAO8evdC4H7LBx3rElxzPOoESJrOSh-L9lKwr4lYJaFihEC9uE5eZZvhePFVCoBRMoaqJK815W6fCJ7OhLV0CXS3eaiUHthvHakZ9VWlg2_BwB06E2htPkyQ0H49WqQCzLbTvYjPluACaNGkMWCG6UfoawzMVYsn_RmRrHlco0LVhAVr9uJOFhnPcOWt5dqeX7ujZ7gxFiFXLRxYRe3ClE-nz8aHaXXbCiBduTbanurpMoUUjVh4C2rvduajK5vD7k_aKHmUKZKGK0eAlFdSY_tQqwdB4flu9RixhNsJhsMBh_FDxR4jZMM0Kw6u89hv0OQKQLXSVXx98Y_SDn1PUb9NiV2JNweHtAPfBAErFTZyAtn9t0SGA9Drx7Mx972a85NloTDECMQnjBL74jiGgs_Hii0_pOk1W79pqHIx8d1T83sXTBLeM_MhBfR53zmLZlJ8Plym_M8NypgOQoezS0nJhtgzz_SJ_A3EEnRvSQ21OhBSqWO7xnXtN64ZxBbfqBJyCW9qcYWM5gR3h7lmTnpqz3nlX6POusBDp7QVLmdfSLKDfdSguWwpAm6A7e7rCt__cwrAqPdUjtsCa9fE4z7t5ppv5UexA2MwvcGYdDls47uQ3O3NWbFFO2A-RJTTQwn00y0A41soZXk4YNhZSUb5we9S_sNSi1N5z2G6g1J921_JTucJ5hifADuPsBoQ2qBmhwYdvmhI2_6MqEQwQCoJa2cW-nWlOra2dqMbUyTPv8C3sQgkcklpEOE0PjmkyL4WUztLe5MSFw7yBQcQ-BCNLYxvti6mKOsmA5iZ_JqFVKmK3lG3nDMP8puHvyDFCXNEL3gLTl2ju-1Dyf2zvv7TPyZs8LKgiV2gtOgw38s_EfBXdiez-DWobTGAlMmkUYljsWWQJwqkOQPLvfPL9KLmKjZlcCmE00mk6Nh4qRcUbHXPRkA-Gdas4WFtfl3MQ3AaHyxswQCXTfzE0Yga5Our7qQf2w9vbw1woNAemk8SeZhDlyhgZ1mjulQ9lz5eeRq0aff2sedsehZBOBH6dtEx0DaGYQe9t9uu1pLeEnTAd8zrpoDjZq4pUd2ElpOx3EptDy96U-Vxo8aQVLy5oK1wThJ-qBMqkJS_ehak57cUbtR_MXBmCEK5N6HbrmxZcJ_FUEKHrDXO9LWMmpnafZPA0kaEyDWM5ZzJ5AUt8N8aX7-s8ft87qdZLA8LnLSjdm4iP1QKV_1pfRJ8cNgFDxR4hFczJ5mBNfH5sGKhJ65AU3Bli8_CeslYw7gEHRCXLri1_-j7m6dJyH397uDcsebW2Npf3j0v8VrrVnwgTJ1S2xzleNbQcmqjCXdeDbxwt9VxBSFklFYdpgd_wk8l536LyMpe3x0iyWbzZ2NZwRnd6z0zq8ALDBK7S_XDXWyVMNc94MrDGzCdzGjVy_ZaDD_uhsnSScxiGN00L5FnUSkqMMC4Xnd80amNIX8e6aO8WOLqeuLKjuiSEkl8QLlt7ZAOeD_Rf8RtPuNwDegMefhnqbdDplYLXDFD--q4qn1H8iwfoutHHN5ZxiL_JmsfmNAXfemsrDw4qgsWCihPTJPBmlet-UY31UrLaR6oqaFpmJgZ2dlm9Pj8GIkvM59zEHN8zNwcV7skG3dm73NWzM0ng3oQoQklazmkWXw8Er23MjqzY0AISYxcte94elLexs3uoMOb0k9DaA397tKxH3YNBLkY9gix6v5Vwjvpp_bSGSEJsRdeC1IFDsavr8VBKAJix-elU61mboEgdepLZEDJgkv3h5AS0u2B5-eVaFCUEIeLoRsIYCJZM3p5G_VZTCwHCZH7NC4Dl7vJ6kzTPhUrsEydjA-87jCgBQ567jW_whiwlOUKIEG0xseSJQ3BrnWNfl3e63NzRCg0y4niducBaS9pTr8Pmp86A
Authentication tag	FFUjIAH_HJEZq7XRuXmV6g
Decrypted payload	7B202276657273696F6E223A2022312E32222C20227369676E6174757265416C676F726974686D223A20225348413235366997468525341222C20227369676E6174757 26554797065223A2022636D732D7061646573222C20227369676E6174757265223A20224D49494B6759496A6B6C33456C4415163436F49999B717A34343371634 341514578447A414E42676C67686B67425A514D454158414146727465446E6893397730424277476767637634949448B7A434342524F6741774942424626749 4542677A4C44414E42676B71686B6947393704277476767763763764D4949448B7A434347213155454367F71355645367F76156604666C633335 276636D5672615584E305A584A706132567A6133567A4946525646553151784744415742674C4431520C63335270646D46796257567564757566CE644445724D43 6B47413155454177777769566C4494526546553151517F5A4C544554614444455443456E794376696873417397A5A584D674C5342448C4441165447E58156604666C635 6d65778736154453486E42441404427435535846547A5F44574F41419594691844969F1D74413B7A4744594459965F4C7D74145659F73063CF75746D7A425415656 5645573617364734554E4413615459146965447677674967594F547241155937D0D3153B764F72656546E546B5A44E4715541594B4E4F5A76D673F65656F566A44E5A E6564A97F0F84B31447F452336527355323632436467394F4B6B7948706736569578970345014576D504A63636F30705097F65094A366366F3072097744C6D703534 355435314332764C76676C466C4F44712B324B426D3966552682F6568474C382E5368B6A66756D6C4A4A57466E4C7369F3D4D65616B65B6E655B45C4A867346533671 6741736A34436B30504B394F69636A366E7867746c72666E466A99936E7864F26A366669936D75946498555F44444457439454316D64B70443A70D0D 8646430737D047C304C324E7737664B354D435456745531B4353A5C6C4C6D6C7A4C4D703166D5567551485221635124587824C4F7B7052E246506D6D2C4536616456516549926B34B325A3665064313 3692F45486A6A7A3353F59495467506566565637495996E464697975351A7476304378396753726947067F7742F412B523541493043565507841524D346500637868D0D 707732767F45635656E367564442566564E5D6D7856436653362235131367073594C6270543655643D6410475534A5686876164793441FE66574647 7963715432483866A6635A707735563151494441514614426F344942566E4416526745447D9C33734757455A415454425E51F674C4159949B7759424251554844 45324573B4C48503B4577485519445E52304F424Z43554494C466B7784C6363369574456538823B3113670773594967263543054B41344F41153564447452F775145 41778947514443427A67594456523067424849494844542067D8A6BB65A4657659596F63963794E434534E455554A46D62B315344B5344F41453648636D316C626D 6337389736158520617577254594667D452B61485953484A2A73DED44776663F61485954842736B552B2B6D4B355568334231644435474762BB758316871335775 41777137576347705645534334536638871059435146B36E45354594F65E644356C43F6462441424614A6E4E5E5461597617364C5B5A584F41153564447452D 7941595944594959449451575564442144342615527354363F76666E5744574A6337697D3437503F547430E466572A50979F757850454425444324F5579536C79 61330635561C3656E4C4F44A9985C67254E4E4884A426477C64764A76F643356E5DD576266742A455E5A6E5641597634D68355C7C1717062675774424432541414D 353435314332764C766D6D30586272749449362615835036366693525822F5658474C858D6136D6531001C4537766E4844636364159596A5D7657D80B3 51697574496C4D2B7F871696E787A302F4C696B64522075534747442723632621477850354234324F5579536C797A2D65514C73656F4C3E4E484F7472344B525661464434 71703635509977135325351674574756454549482E6570334F065E5626F44E47526C3931201435F58467E44E43266352E617D330306D5361517E14837A69766E44C574 D493072733432516560F4C543067472832219F615547367655617175A7A7047E4A7A5A4C4F4A34A5486C6D6153752B522F50553074704C314B7361797F5A7570714A7 78475946574766572537A4E4838552F6698795A4E85306944272147162466B5844584047ED493581327864637430399474F30C71435374C6C7A7 375A76393632B7A71536A736D6A2A73516A5F796647666C25A574F44E465A3538394733572B786A5642766B36F487831460F764C748843336A6137456945156372F74 5A9BF74563D4394A17F02427654F53537B5A426A5D3765D95A6C74448F5A2D66E7B4C4B7513155454288A65454F65642416F4D476C5A6885B4E4D E63367A44445727D8A6644671774691574764743366465326953104E6524731766645449E9704344F44A44A44E1315455464424702B54704B54B41453D766EF 68854564726D64C56623646A5C71485240A31423437366C4549009704374349174534496242F79436C4A70A79754E466E56F444A2E44C60D74B44E44FE 6874A544744A54394A5459595A4941157555444241494425520143676744534567A69546C475644574A4A3F62F62654642F6E45749493370720C724F73A6E5B B6F5A496876634E41516B454C5A94954945B4437557562F386678547553342067541567465966833163C33834C55736B6F395A334B5726430674153625A420F7304049854149724 68686947373042435241143404C7A4742774443427654543424324756A4343427ADAA4F441C41C426762C7C8663B667425A51DF4547645459506226297D42770A387670830356F4744374381515A6 E655117478B65624138554D626B48141597761794E58629354D4947446D3826568D5731774531594455651514757774A47355456A4D434547413155545467 61566E44577241155BF45C4A9E52A75B6663774B41167B547443A354747B54544743F65647794446D6737A24354D5137374B155F45F65155571FF74441554A76674525341597375A5A4844F44357E36E1315455454A63757 66C64444572D4D3641314154577F654A4967B446977974B69C64F76A83706F764A7365E4A6E576D64D46A56F0966A0F5C42D664A45C7 66E967584B34317A537A754577333954357A476F694533A4306E4536E426F35796F63535155F1566A6B704654B85B7275F6A276F6F694547B6667950 549684A694341784D7951326736734D9634524357373577F45744766537703574E6667A34A379636C5934442426D704F62644552743626C74514C466443B67C7A6267 45537037515A7473633650505862772774943626135030395439726532DE2D2B78535465364447835530617342D3254424D6F4415269355936716567352663744494D34875 71487034706559967939724C48685844774766363653869F44446F2F534C2B5374F44519617878795037647838356E6C31572626433886642A5555 5325654474E326A62863484F6A49525577455567258456737557939624B752F654963466B36666E5A7132342A6B51316F16694940C56444848974347552563636D7C 6B67364A79754F70784951783967515693834501677194B364F4C33767442725253674644A4355734576E756978627036A46B8706875275375374615641663D34464 A44F1357B33323222020273746117475173232A0226F6B222C2022726561736C736F4E6465227A20223300223020C202027265726F6E6E436C736436467A5A0225369E695651 67656E6572261746672077
Plaintext payload	{ "version": "1.2", "signatureAlgorithm": "SHA256withRSA", "signatureType": "cms-pades", "signature": "MIIKugYJKoZIhvcNAQcCoIIKqzCCCqcCAQExDzANBglghkgBZQMEAgEFADALBgkqhkiG9w0BBwGgggcvMIIHKzCCBROgAwIBAgIEBgzGIDANBgkqhkiG9w0BAQ0FADB5 MQswCQYDVQQGEwJGSTEjMCEGA1UECgwaVmFlc3RvcmVraXN0ZXJpa2Vza3VzMD1jRcvmVraNVn0ZXJpa2Vza3VzYtRFU1QwH01AZglqU7uMSbSMLLLDEr MCk9GA1UEAwwYVJLIFR FU1QgQ0EgZm9yIFRlc3QgUHvycG9zZXMgLSBHNDAeFw0xOTA1MjAxMTI4MjNaFw0yNDA yMjUyMTU5N TlaMH0xCzAJBgNVBAYTAkZJMQ4wDAYDVQQKDAVUdWxsxaT ESMBAGA1UEBRMJOTk5MDQ5NzEwEQYDVQQDAVERU5JUzEUMBIGA1UEBAwLSIVVEIMQUIORU4xJDAiBgNVBAMMG0pvVVFJTEFJ TkVoIVRFF TkITjJKtVOIERFTkiTIDk5OT A0OTcxMDCCAAlwDQYJKoZIhvcNAQEBBQADggEPADCCAYoCggEBAMhYM0U3szUviDon/lLDugHLosOccs8hhbF8wiKRRz/LHpVY4Ys3vlum2cc4fz1hLg30hkZD50Xxicq H92WCVkmw/6nXei6nJzhNiu7AyX+fnZxgHCRpdykcxuUgcxBc4lXnDppkdUb.bvbvF5RaXkJxpnkgCUnMK2aSQXCmCh0/nSfDDq3ocBEBhpJ8zQ+rSCGdgGbupegGwEN3c zHXbLNrO9gf1Ul1dgAsj4Ck0PK9Ocj6nxgGLbfj9f8yDnEVfIaGaPiaTB95+1FAleTsMT8YSYAc27CwFcHhdd01pLqb8jsDh9MAs1h92qtmLGCe2XwTk4qn194acDCwngeeOywk RbAxlGgv7mXml13i/EHjjz3SoYlToPfeV7lYnFlyu5ztv0Cx9gSriCpowM/A+R5Al0CVUPxARM3FPcvn09pw2rweEcUkN6ujdBPo4Z5y+HQbkVxw1rCSywcWGqUgP0QhTVMd4Oi rcZEhvQdy3ONfWEGycqT2H8jf7Zpw5V1QIDAQABo4CANTCCAjEwHwYDVR0jBBgwFoAUPZqitfgVEe8RyuvHXE2TgLLHP8EwHQYDVR00BBYEFLFkwxLc3eyEe8b3Q16ps YLbpT5eMA4GA1UdDwEB/wQEAwIGQDCBzgYDVR0gBIHGMIHDMIHABgoqgXaEBWMKgUsBMIGxMGcGCCsGAQUFBwIBFhtodHRwOi8vd3d3.lmZpbmVpZC5maS9jcHM 5OS8wgYUGCCsGAQUFBwICMHkad1Zhcm1lbm5cG9saXRpaklrYSByb2xY3XZpbGxhlCd2QgGlmaWtdBwb2xyY3kgZmlubnMgLSBDZXJ0aWmpZ25zaWdrY2F0aW9u2xpY3kgc2FBpbnhy2xz 2xpY3kgaXMgYXZhaWxhYmxlIGh0dHA6Ly93d3cuZmluZWlkLmZpL20zpLm5wcz5MCUGA1UdEQQeMByBGmRlbmizLmp1dXRpbGFpbmVuQR1bpxpmZmM8GA1UdEwEB/ wQFMAMBAQAwNwYDVR0fBDAwLjAsoCqgKlYmaHR0cDovL3Byb3h5LmZpbmVpZC5maS9jcmwvdrJdHA0Yy5jcmwwbgYIKwYBBQUHAQEEYjBgMDAGCCsGAQUFBzA ChiRcodHRwOi8vcHJveHkuZmluZWlkLmZpL2NhL3Zya3RwNC5jcnQwLAYIKwYBBQUHMAGGIGh0dHA6Ly8vd2dc5maW5laWQuZmkvcHJvdHA0MC0wGCCsGAQU FBwEDBCEwHzAJBgYEAISGAQEwEwYYGBAC0RgEGMAkGBwQAjkYBBgEwDQYJKoZIhvcNAQEBBQADggEBAD7+WAav3WfDY2N3WliXtgpSwmE/aHYSHBkkU++mK5Uk 3B1dE4WGa+ku1hg3WvKNQ6PtGq7WcGWpdU4$V8hpYCQKcTJevYYoc9WkN4ZD6EncFJ/zS00/cOZRQlwlxdo5EmyYicWyANfKJt9PLIVb6ELqKRG1vlEIipCJrRgFPcJj+Cv WsxZ1d8N02nKajmGpG443x7J7oLxXNWOgBS6RsU262Cdn9OKkyKpg5iXip4PygyPUJ6cco0pYwD+2zqpbmuwDG2QLGGi+Y5T51C2vLvgIFlKKq+2KBm9fRX/ehGN8Shj7eo v1MePAE7vntMCa4FY2h9h7U26+Z6cRQiuYIlM+xqinxz0/LikdRqu4GDr62bQGxP5B42OUySlyzMeQLseoL7NHOtr4KRVaFD3znL2fPtLh6ISSSHFPoEn4bfWP9oh4 NGRk0ukcFLrltEpixuQr+8eYDpmk1dMl0rs42QelT0gBx2A9/aUG6vVqqZzpONzZLOJ9JEHlmASu+R/PSPtpL1KsaywZupqJWxGYNvtivXrSzNK6R/ikyZNkZ0IDRtqbFkkD8jpO ml5a2xdczl0f707rxJLiX8TD5Cc/7jv96+zqSjsmj2zsQkitflZ/OOrcSgzec589G3W+xjVBvkSoHx1mLthC3ja7EiEV7/tA/4vmdobC9KEMYIDTzCCA0sCAQEwgYEweTELMAkGA1U

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

EBhMCRkkxIzAhBgNVBAoMGlZhZXN0b3Jla2IzdGVyaWtc2t1cyBURVNUMRgwFgYDVQQLDA9UZXN0aXZhcm1IbnRIZXQxKzApBgNVBAMMIIZSSyBURVNUIENBIGZvciB
UZXN0IFB1cnBvc2VzIC0gRzQCBAYMxpQwDQYJYIZIAWUDBAIBBQCgggEeMBgGCSqGSIb3DQEJAzELBgkqhkiG9w0BBwEwLwYJKoZIhvcNAQkEMSIEIH+DsWV/8fxTu
S3BgUih1I38LUsfo9Z3KErd0gASbZBpMIHQBgsqhkiG9w0BCRACLzGBwDCBvTCBujCBtzALBglghkgBZQMEAgEEIPbimrpK6yqx05nCqQ+nfQtxkebA8UMbkHAYwayNXb9T
MIGFMH2kezB5MQswCQYDVQQGEwJGSTEjMCEGA1UECgwaVmFlc3RvcmVraXN0ZXJpa2Vza3VzMR0wGwYDVQQLDBRDANVBgNVBAsMD1RIc3RpdmFybWWvudGVIdDErMCkGA
1UEAwwiVIJLIFRFU1QgQ0EgZm9yIFRIc3QgUHVycG9zZXMgLSBHNAIEBgzGtDANBgkqhkiG9w0BAQEFAASCAYCsJ1jZ9GQgExfvLX1CZXaQCut7GcO3qICuyjOCgeik41z
SzuEw39T5zGoiE5MF5nBo5ybUSQVjkpFTXkBG0dqxuZWUU/mw/iGyG4Iupk9QuIhJiCAxMyQ2g6zv8DRCS3ywEGFuc7p5utnFnz0yclY4BMBkpObd7GuCbatQLdMCIz2gES
p7QZts6PPXbrwICba5P9T9re0+xUE6Dx50asM2TBMoARi5YaVsRftDlMsAHo8dEfIwqHp4peYny8my2LHhXDwiccXito/SL+St1EIaxxyP7dx85nl1ErnC8IlyP6rHdYvkJuUU2VTG
N2jbhcHOjIRUwEVrXEg7Uy9bKu/eIcFk6fnZq3BZkQ1oafILVDHKy4uRV8+HTLFkg6JyuOpxIQx9gQiX4P6wyk6OL3vtBrRS6zJ5W4Wnuixbp8NKphurustiaUAfBsSME2PZ0m96
QS5renB5XorViR/lMABJZ5XLVqvEC+IpgvpiFuVUBzdQE/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.:

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

		- 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"
}
```

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

3   Security considerations

This section is non-normative.

3.1   Server certificate validation

Before signing a document using a smart card, the signer should always verify the authenticity and validity of the requestor's certificate. This crucial step helps prevent fraudulent activities and ensures the signer is interacting with the intended recipient.

Verification confirms that the certificate hasn't been revoked, ensuring the recipient's identity, and preventing the signer from unknowingly signing a document intended for a malicious actor.

Failing to verify the certificate exposes the signer to risks such as signing documents they didn't intend to, potentially leading to identity theft, legal, or financial liabilities. Therefore, verifying the certificate is a fundamental security practice that protects both the signer and the integrity of the digital signature.

3.2   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.

In addition, the client application SHALL use only the "signature" as "signatureType". Other types do not validate the origin and do not add a "proof of origin" when using transaction-based signatures. See chapter 2.1 for more information on how to generate an proper authentication request.

3.3   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)

3.4   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 3.10 on Malware.

3.5   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

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

SCS will always show the certificate selection dialog when a signature creation request is received.

3.6   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.

3.7   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.

3.8   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.

3.9   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.

3.10   Malware

As the SCS exposes a server socket on localhost (see section 3.4), 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.

3.11 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.

3.12 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: <u>GET</u>, <u>POST</u>

HTTP URI: <u>/sign</u>, <u>/version</u>

Hash algorithms: "<u>SHA256</u>" (default), "SHA384", "SHA512"

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

4.2   RSA signature profile

Extends Base profile.

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

Signature types: "<u>signature</u>"

4.3   ECDSA profile

Extends Base profile.

Signature algorithms: "<u>SHA256withECDSA</u>" (default), "SHA384withECDSA", "SHA512withECDSA"

Signature types: "<u>signature</u>","signature-plain"

4.4   CMS signature profile

Extends Base profile.

Signature types: "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):

  o contentType: The same value as in contentInfo.contentType field

  o messageDigest: If the SCS request contentType is "data", digest is computed from the given content by using the digest algorithm specified in signerInfo.digestAlgorithm. If contentType is "digest", content is used as it was given by the requester directly.

  o signingTime: The time when this CMS object was signed

- signerInfo.digestEncryptionAlgorithm:

  o When using RSA key:

    ▪ PKCS#1 rsaEncryption object identifier

- When using ECC key:

  o ECDSA signature algorithm identifier, SHA256withECDSA, SHA384withECDSA, or SHA512withECDSA

Kehitys ja tietohallinto / Jari Pirinen   1.3.2025

- 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.

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

- signerInfo.authenticatedAttributes.signingTime

- signerInfo.digestEncryptionAlgorithm (=digestAlgorithm)