Smart-ID Relying Party integration guide and API specification

- 1. Introduction
  - 1.1. Terminology
- 2. References
- 3. General description
  - 3.1. UUID encoding
  - 3.2. Hash algorithms
  - 3.3. Relying Party authentication
  - 3.4. API endpoint authentication
- 4. REST API
  - 4.1. Interface patterns
    - 4.1.1. Session management
    - 4.1.2. REST object references
    - 4.1.3. HTTP status code usage
    - 4.1.4. Response on successful session creation
    - 4.1.5. Idempotent behaviour
  - 4.2. REST API main flows
  - 4.3. Certificate choice session
    - 4.3.1. Preconditions
    - 4.3.2. Postconditions
    - 4.3.3. Error conditions
    - 4.3.4. Request parameters
    - 4.3.5. Example response
  - 4.4. Authentication session
    - 4.4.1. Preconditions
    - 4.4.2. Postconditions
    - 4.4.3. Error conditions
    - 4.4.4. Authentication request parameters
    - 4.4.5. Example response
  - 4.5. Signing session
    - 4.5.1. Preconditions
    - 4.5.2. Postconditions
    - 4.5.3. Error conditions
    - 4.5.4. Request parameters
    - 4.5.5. Example response
  - 4.6. Session status
    - 4.6.1. Preconditions
    - 4.6.2. Postconditions
    - 4.6.3. Error conditions
    - 4.6.4. Response structure
- 5. Session end result codes
- 6. Protocols
  - 6.1. Authentication protocol
    - 6.1.1. Sending authentication request
    - 6.1.2. Computing the verification code
    - 6.1.3. Verifying the authentication response

Date	Version	Change description	Author
28.09.2016	0.92	Better examples in REST API. **rpName** renamed to **relyingPartyName** in REST API.	Mart Oruaas
14.10.2016	1.0	Clarified and changed certificate and assurance level usage in REST API.	Mart Oruaas
27.10.2016	1.1	Removed JSON-RPC API description	Mart Oruaas
03.11.2016	1.2	Added optional request nonces	Mart Oruaas

1. Introduction

Relying Party interface offers the entry point to Smart-ID main use cases, i.e. authentication and signing.

The interface is to be used by all parties who wish consume Smart-ID services, i.e. ask end users to perform authentication and signing operations.

Cybernetica AS reference: Y-952-4, version 1.2.

1.1. Terminology

- **Document number** - A unique number given to end user accounts in Smart-ID system, identifies a particular set of end user cryptographic key pairs with their certificates.
- **Relying Party (RP)** - a provider of some kind of e-service (like an internet banking site or government agency website), or, technically, that same e-service. Relying Party authenticates users via Smart-ID service and requests digital certificates from them.
- **Session** - Alternatively, "RP Request". A process initated by Relying Party, which contains a single certificate choice, authentication or signing operation.

2. References

1. R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Internet Engineering Task Force, June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt.

3. General description

Relying Party API is exposed over a REST interface as described below.

All messages are encoded using UTF-8.

3.1. UUID encoding

UUID values are encoded as strings containing hexadecimal digits, in canonical 8-4-4-4-12 format, i.e. `de305d54-75b4-431b-adb2-eb6b9e546014`.

Smart-ID uses version 4 (random) UUID values.

3.2. Hash algorithms

Smart-ID supports signature operations based on SHA-2 family of hash algorithms, namely SHA-256, SHA-384 and SHA-512. Their corresponding identifiers in APIs are "SHA256", "SHA384" and "SHA512".

3.3. Relying Party authentication

Interface users are authenticated based on their originating IP-address and `relyingPartyUUID` protocol parameter combinations.

When authentication fails, server must respond with HTTP error 401.

3.4. API endpoint authentication

It is essential that RP performs all the required checks when connecting to the HTTPS API endpoint, to make sure that the connection endpoint is authentic and that the connection is secure. This is required to prevent MITM attacks for the authentication and signature protocols.

The RP must do the following checks:

1. Verify if the HTTPS connection and the TLS handshake is performed with the secure TLS ciphersuite.
2. Verify that the X.509 certificate of the HTTPS endpoint belongs to the well-known public key of the Smart-ID API. The RP must implement HTTPS pinning (https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)
3. Verify that the X.509 certificate of the HTTPS endpoint is valid (not expired, signed by trusted CA and not revoked)

In case the RP fails to verify the connection security and the attacks is able to launch MITM attack (https://en.wikipedia.org/wiki/Man-in-the-middle_attack) on the connection and circumvent the connection authentication, the following attack is possible:

1. End user connects to the RP website and asks for the authentication with the Smart-ID.
2. RP connects to the authentication API endpoint, but attacker is able to MITM the connection and answer himself.
3. RP sends the correctly formed authentication request with randomly generated hash (h1) to the attacker, acting as the Smart-ID API.
4. Attacker creates another connection to the RP website and asks for the authentication with the Smart-ID, under the identity of same end user.

5. RP connects to the authentication API endpoint, but attacker is able MITM the connection and answer himself.
6. RP sends the correctly formed authentication request with randomly generated hash (h2) to the attacker, acting as the Smart-ID API.
7. Attacker computes the VC values for both hashes, i.e.
    a. vc1 = integer(SHA256(h1)[2:1]) mod 10000
    b. vc2 = integer(SHA256(h2)[2:1]) mod 10000
8. If the vc1 != vc1, the attacker drops the connection to the RP website and creates a new one. The connections are tried until the randomly generated hash value yields the same VC value as the vc1. It should take about 5000 tries, before such collision is found.
9. The attacker sends the authentication request with the hash value h2 to the Smart-ID authentication API endpoint.
10. Smart-ID sends the authentication request to the end user mobile device and asks to verify the VC.
11. End user compares the vc1 displayed on the browser to the vc2 displayed on the mobile device and finds that they are equivalent and consents to the authentication.
12. Attacker receives the authentication response from the Smart-ID API and returns this to the RP connected, associated with the attacker's session.
13. RP receives the authentication response with the signature on the hash h2, verifies that the signature is valid and creates authenticated session for the attacker, under the end user identity.
14. The attacker is logged in as the end user.

4. REST API

4.1. Interface patterns

BASE URL value is given in service documentation and may vary between environments and service instances.
The base URL for SK DEMO environment is https://sid.demo.sk.ee/smart-id-rp/v1/

4.1.1. Session management

Base of all operations on the API is a "session".

Session is created using one of the POST requests and it ends when a result gets created or when session ends with an error.

Session is identified by an ID, in reality a long random string.

Session is created for one of the three operations:

- Authentication
- Signing certificate choice (needed for certain digital signature schemes, see below)
- Signing

Session result can be obtained using a GET request described below.

4.1.2. REST object references

- Objects referenced by "pno/:country/:national-identity-number" are natural persons identified by their ETSI PNO-type identifier (i.e. PNOEE-12345 becomes pno/EE/12345)
- Objects referenced by "document/:documentnumber" are particular documents (also known as user accounts) in the Smart-ID system.

4.1.3. HTTP status code usage

Normally, all positive responses are given using HTTP status code "200 OK".

In some cases, 4xx series error codes are used, those cases are described per request.

All 5xx series error codes indicate some kind of fatal server error.

4.1.4. Response on successful session creation

This response is returned from all POST method calls that create a new session.

Parameter	Type	Mandatory	Description
sessionID	string	+	A string that can be used to request operation result, see below.

successful session creation response

```
{
	"sessionID": "de305d54-75b4-431b-adb2-eb6b9e546014"
}
```

4.1.5. Idempotent behaviour

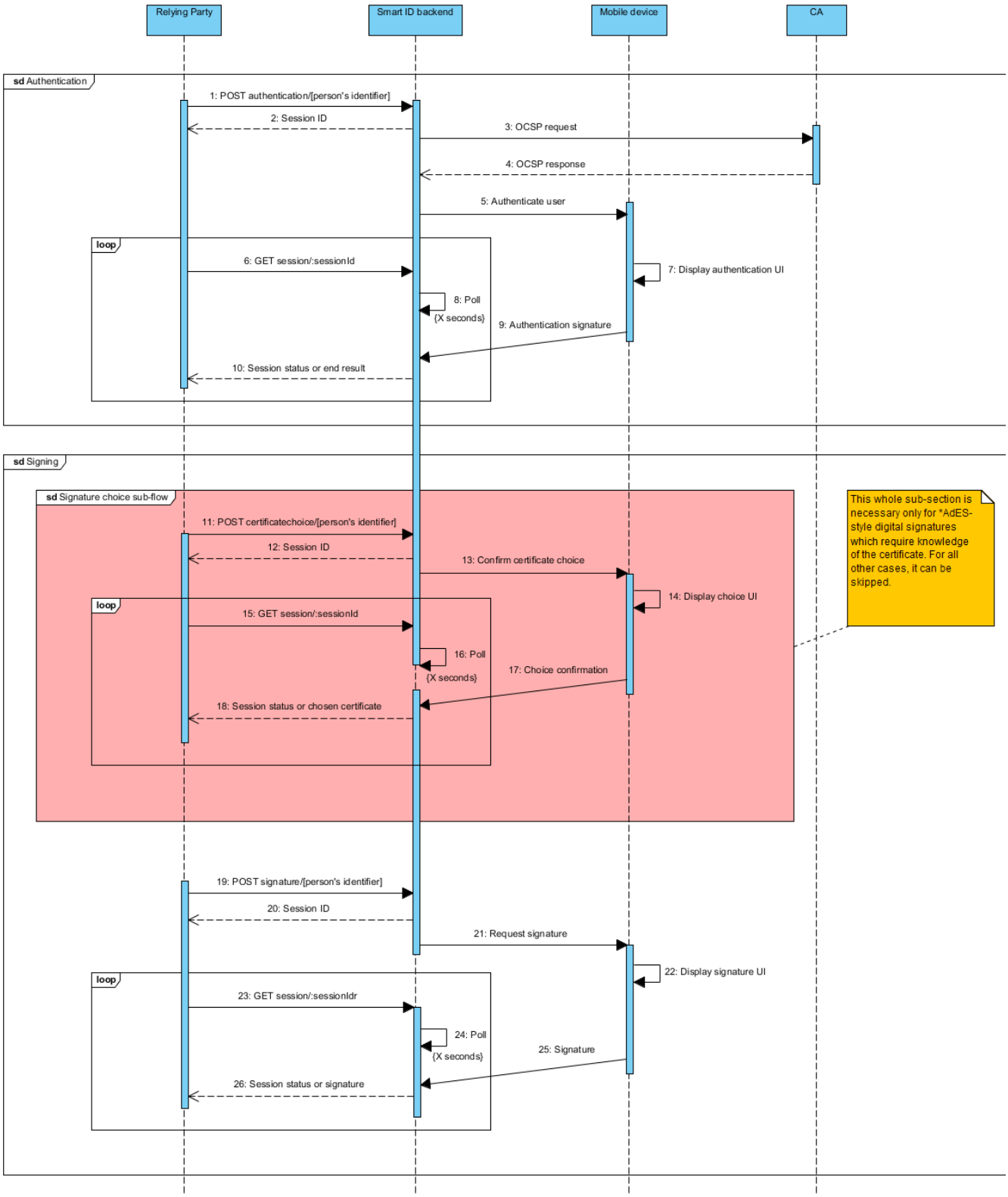
Whenever a RP session creation request (POST to certificatechoice/, signature/, authentication/) is repeated inside a given timeframe with exactly the same parameters, session ID of an existing session can be returned as a result.

This allows to retry RP POST requests in case of communication errors.

Retry timeframe is **15 seconds**.

When requestor wants, it can override the idempotent behaviour inside of this timeframe using an optional "nonce" parameter present for all POST requests. Normally, that parameter can be omitted.

4.2. REST API main flows



4.3. Certificate choice session

Method	URL
POST	BASE/certificatechoice/pno/:country/:national-identity-number
POST	BASE/certificatechoice/document/:documentnumber

Initiates certificate choice between multiple signing certificates the user may hold on his/her different mobile devices.

Having a correct certificate is needed for giving signatures under *AdES schemes.

This method can be ignored if the signature scheme does not mandate presence of certificate under the signature itself.

This method initiates a certificate (device) choice dialogue on end user's device, so it may not be called without explicit need (i.e. it may be called only as the first step in signing process).

4.3.1. Preconditions

- User identified in the request (either by PNO identifier or document number) is present in the system.
- User has certificate(s) with level which is equal to or higher than the level requested.

4.3.2. Postconditions

- New session has been created in the system and its ID returned to Relying Party.

4.3.3. Error conditions

- HTTP error code 404 - object described in URL was not found, essentially meaning that the user does not have account in Smart-ID system.

4.3.4. Request parameters

Parameter	Type	Mandatory	Description
relyingPartyUUID	string	+	UUID of Relying Party
relyingPartyName	string	+	RP friendly name, one of those configured for particular RP
certificateLevel	string	+	Level of certificate requested. "ADVANCED"/"QUALIFIED"
nonce	string		Random string, up to 30 characters. If present, must have at least 1 character.

Certificate choice request

```
{
 "relyingPartyUUID": "de305d54-75b4-431b-adb2-eb6b9e546014",
 "relyingPartyName": "BANK123",
 "certificateLevel": "ADVANCED"
}
```

4.3.5. Example response

Certificate choice session creation response

```
{
       "sessionID": "de305d54-75b4-431b-adb2-eb6b9e546014"
}
```

4.4. Authentication session

Method	URL

POST	BASE/authentication/pno/:country/:national-identity-number
POST	BASE/authentication/document/:documentnumber

This method is the main entry point to authentication logic.

It selects user's authentication key as the one to be used in the process.

4.4.1. Preconditions

- User identified in the request (either by PNO identifier or document number) is present in the system.
- User has at least one account with given assurance level.

4.4.2. Postconditions

- New session has been created in the system and its ID returned to Relying Party.

4.4.3. Error conditions

- HTTP error code 404 - object described in URL was not found, essentially meaning that the user does not have account in Smart-ID system.

4.4.4. Authentication request parameters

Parameter	Type	Mandatory	Description
relyingPartyUUID	string	+	UUID of Relying Party
relyingPartyName	string	+	RP friendly name, one of those configured for particular RP

assuranceLevel	string	+	Level of assurance requested:
			`"http://eidas.europa.eu/LoA/low"`
			`"http://eidas.europa.eu/LoA/substantial"`
			`"http://eidas.europa.eu/LoA/high"`

hash	string	+	Base64 encoded hash function output to be signed.
hashType	string	+	Hash algorithm. See hash algorithm section, but authentication is limited to SHA512.
displayText	string		Text to display for authentication consent dialog on the mobile device
nonce	string		Random string, up to 30 characters. If present, must have at least 1 character.

Authentication request

```
{
   "relyingPartyUUID": "de305d54-75b4-431b-adb2-eb6b9e546014",
   "relyingPartyName": "BANK123",
   "assuranceLevel": "http://eidas.europa.eu/LoA/high"
   "hash": "ZHNmYmhkZmdoZGcgZmRmMTM0NTM...",
   "hashType": "SHA512",
   "displayText": "Log into internet banking system"
}
```

4.4.5. Example response

Authentication session creation response

```
{
        "sessionID": "de305d54-75b4-431b-adb2-eb6b9e546015"
}
```

4.5. Signing session

Method	URL
POST	BASE/signature/pno/:country/:national-identity-number
POST	BASE/signature/document/:documentnumber

This method is the main entry point to signature logic.

4.5.1. Preconditions

- User identified in the request (either by PNO identifier or document number) is present in the system.
- RP knows the user's signing certificate related to particatular document, if needed by signature scheme, the related certificate.

4.5.2. Postconditions

- A new RP request request and a related transaction record has been created.

4.5.3. Error conditions

- HTTP error code 404 - object described in URL was not found, essentially meaning that the user does not have account in Smart-ID system.

4.5.4. Request parameters

Parameter	Type	Mandatory	Description
relyingPartyUUID	string	+	UUID of Relying Party
relyingPartyName	string	+	RP friendly name, one of those configured for particular RP
certificateLevel	string	+	Level of certificate requested. "ADVANCED"/"QUALIFIED"
hash	string	+	Base64 encoded hash function output to be signed.
hashType	string	+	Hash algorithm. See hash algorithm section.
displayText	string		Text to display for signature consent dialog on the mobile device
nonce	string		Random string, up to 30 characters. If present, must have at least 1 character.

Signature request

```
{
   "relyingPartyUUID": "de305d54-75b4-431b-adb2-eb6b9e546014",
   "relyingPartyName": "BANK123",
   "certificateLevel": "ADVANCED"
   "hash": "ZHNmYmhkZmdoZGcgZmRmMTM0NTM...",
   "hashType": "SHA512",
   "displayText": "Authorize transfer of £10"
}
```

4.5.5. Example response

Signature session creation response

```
{
      "sessionID": "de305d54-75b4-431b-adb2-eb6b9e546016"
}
```

4.6. Session status

Method	URL
GET	BASE/session/:sessionId

Query parameter	Contents
timeoutMs	Request long poll timeout value in milliseconds. If not provided, a default is used.

This method can be used to retrieve session result from Smart-ID backend.

This is a long poll method, meaning it might not return until a timeout expires. Caller can tune the request parameters inside the bounds set by service operator.

Example URL:

`https://sid.demo.sk.ee/smart-id-rp/v1/session/de305d54-75b4-431b-adb2-eb6b9e546016?timeoutMs=10000`

4.6.1. Preconditions

- Session is present in the system and the request is either running or has been completed less than 5 minutes ago.

4.6.2. Postconditions

- Request result has been returned to RP.

4.6.3. Error conditions

- HTTP error code 404 - session does not exist or is too old or has expired.

4.6.4. Response structure

Parameter	Type	Mandatory	Description
state	string	+	State of request. "RUNNING"/"COMPLETE". There are only two possible status codes for now.
result	object		Structure describing end result, may be empty or missing when still running
result.endResult	string	+	End result of the transaction. Refer to the subsection below.
result.documentNumber	string	for OK	Document number, can be used in further signature and authentication requests to target the same device.
signature	object		Structure describing the signature result, if any.
signature.value	string	+	Signature value, base64 encoded.
signature.algorithm	string	+	Signature algorithm, in the form of sha256WithRSAEncryption
cert	object	for OK	Structure describing the certificate related to request.
cert.value	string	+	Certificate value, DER+Base64 encoded.
cert.ocspResponse	string		Fresh OCSP response for the certificate, DER+Base64 encoded. Present only for authentication response.
cert.assuranceLevel	string		Level of assurance offered during authentication, equal to or higher than the one requested. "LOW"/"SUBSTANTIAL"/"HIGH" **TODO: may move during auth. protocol enchancements**
cert.certificateLevel	string		Level of signing certificate, equal to or higher than the one requested. "QUALIFIED"/"NONQUALIFIED".

successful response when still waiting for user's response

```
{
    "state": "RUNNING",
    "result": {}
}
```

successful response after completion

```
{
    "state": "COMPLETE",
 "result": {
            "endResult": "OK",
            "documentNumber": "PNOEE-372...."
    },
    "signature": {
        "value": "B+C9XVjIAZnCHH9vfBSv...",
        "algorithm": "sha512WithRSAEncryption"
    },
    "cert": {
        "value": "B+C9XVjIAZnCHH9vfBSv...",
        "ocspResponse": "B+C9XVjIAZnCHH9vfBSv...",
        "assuranceLevel": "HIGH"
    }
}
```

5. Session end result codes

- **OK** - session was completed successfully, there is a certificate, document number and possibly signature in return structure.
- **USER_REFUSED** - user refused the session.
- **TIMEOUT** - there was a timeout, i.e. end user did not confirm or refuse the operation within given timeframe.
- **DOCUMENT_UNUSABLE** - for some reason, this RP request cannot be completed. User must either check his/her Smart-ID mobile application or turn to customer support for getting the exact reason.

6. Protocols

Previous sections give the overview of the specific API methods, which can be used to perform the authentication and signature operations. This section gives the overview of how to securely combine them and how to deduce the operation result.

6.1. Authentication protocol

6.1.1. Sending authentication request

The RP must create the new hash value for each new authentication request. The recommended way of doing this is to use the Java SecureRandom class (https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html) or equivalent method in other programming languages, to generated random value for each new authentication request.

For example, the following code snippet generates the 64 random bytes and computes the hash value and encodes it in Base64.

```
SecureRandom random = new SecureRandom();
    byte randBytes[] = new byte[64];
    random.nextBytes(randBytes);
 MessageDigest md = MessageDigest.getInstance("SHA-512");
 md.update(randBytes);
 byte[] hash = md.digest();
 byte[] encodedHash = Base64.encodeBase64(hash);
```

The value of the 'encodedHash' must be recorded in the current user's session for later comparison.

6.1.2. Computing the verification code

The RP must then compute the verification code for this authentication request, so that user can bind together the session on the browser and the authentication request on the Smart-ID app. The VC is computed as

```
integer(SHA256(hash)[2:1]) mod 10000
```

where we take SHA256 result, extract 2 rightmost bytes from it, interpret them as a big-endian unsigned integer and take the last 4 digits in decimal for display. SHA256 is always used here, no matter what was the algorithm used to calculate hash.

Please mind that hash is real hash byte value (for example, the byte array returned from the md.digest() call), not Base64 form used for transport.

The VC value must be displayed to the user in the browser together with a message that ask the end user to verify the code.

6.1.3. Verifying the authentication response

After receiving the transaction response from the getRequestResult() API call, the following algorithm must be used to decide, if the authentication result is trustworthy and what is the identity of the authentication end user.

1. "result.endResult" has the value “OK"
2. “signature.value” is the valid signature over the same “hash”, which was submitted by the RP.
3. “signature.value” is the valid signature, verifiable with the public key inside the certificate of the user, given in the field “cert.value"
4. The person’s certificate given in the “cert.value” is valid (not expired, signed by trusted CA, with correct OCSP response in the field “cert.ocspResponse” and with correct assuranceLevel).

5. The identity of the authenticated person is in the 'subject' field of the included X.509 certificate.

After successful authentication, the RP must invalidate the old user's browser or API session identifier and generate a new one.