JavaScript Library for Signing in Web Browsers

Document Version: 1.2

Library Version: 0.21

Last update: 11.02.2015

1. Document versions

Document information	
Created on	01.04.2010
Reference	JavaScript library for Signing in Web Browsers
Receiver	Sertifitseerimiskeskus AS
Authors	Ahto Jaago, Urmo Keskel, Kristi Uukkivi
Version	1.2

Version information		
Date	**Version**	**Changes**
05.06.2014	1.0	Document updated, restructured and translated to English. Made changes according to library's version 0.20, added information according to changes between versions 0.14 and 0.20 of the library.
19.09.2014	1.1	Added notification to chap. 4 about the usage restrictions of asynchronous functions.
11.02.2015	1.2	Updated library's version number.

Table of contents

2. Introduction

This document describes **JavaScript library (idCard.js)** that simplifies implementing signature creation in web applications with **Electronic Identity Cards (eID)** or other PKI-enabled tokens. The library offers JavaScript API which enables communication with the signer's token in web applications (i.e. browser environment).

To achieve the above mentioned, the idCard.js library uses **browser signing module** (plug-in or extension depending on platform) as a base component. The library acts as a bridge between the web application and the signing module. The browser signing module is a component that is installed in the browser, it is used to forward the data from the browser environment to the signer's token.

The idCard.js library along with sample code and documentation can be accessed from http://id.ee/index.php?id=30496.

The idCard.js library (along with the browser signing module) is supported in **Chrome, Mozilla Firefox, Internet Explorer** and **Safari web browsers** in Windows, Mac OS and Ubuntu environments. The library has been tested with eID cards like Estonian, Finnish, Latvian and Lithuanian ID-cards and Estonian Digi-ID.

This document covers the following information:

- Section 3 gives overview of components that are involved in signature creation in a web application with idCard.js library. The library's functional properties, dependencies and prerequisites and the browser signing module are also described.
- Section 4 covers the idCard.js library's API methods and objects, including samples for implementation
- Sections 5 and 6 define terms and acronyms and provide links to additional resources.
- Appendix 1 provides a sample signature creation process diagram

NB! For the users of the idCard.js library's version prior to v0.20 – starting from version 0.20 of the library, the IdCardPluginHandler object's methods' API and usage principles have been changed. Users of version 0.14 (or earlier) need to implement the following changes:

- It is required to implement methods which enable to **asynchronously** request the results of getCertificate(), sign() and getVersion() methods;
- The idCard.js library's getCertificate(), sign() and getVersion() methods need additional input parameters according to the asynchronous methods implemented by the user (as described in the previous point).

Detailed description of the changes are described in chapter "4.2.1 Object IdCardPluginHandler(lang)"

3. Overview

3.1 idCard.js library's functional properties

idCard.js library ([1]) enables using the following functionality in web applications:

1. loading the browser signing module to a web page
2. reading the signer's certificate from signature token (e.g. smart card)
3. signing data with signature token, i.e. sending the data to be signed to the signature token (eID card or other PKI enabled token) and receiving the signature value calculated with the signer's private key.

The library supports **browser signing module** (see also chap. 3.4) which is used for communicating with the signature token in the user's system. The library provides a common API for using different versions of the browser signing module.

3.1.1 Tokens

idCard.js and the browser signing module potentially support all PKI-enabled tokens provided that there is a driver installed in the user's system for communicating with the token in lower level.

The following eID cards have been tested with idCard.js and browser signing module:

- Estonian national ID-card;
- Estonian Digi-ID card;
- Finnish national ID-card (FinEID);
- Latvian national ID-card;
- Lithuanian national ID-card.

3.1.2 User environment

idCard.js is platform-independent, the library and browser signing module are officially supported (i.e. periodically tested) in the following environments:

	Windows	Mac OS	Ubuntu
Google Chrome	+	+	+
Mozilla Firefox	+	+	+
Internet Explorer	+	-	-
Safari	-	+	-

3.2 Dependencies and prerequisites

Before using the idCard.js library, the following prerequisites have to be fulfilled:

1. The signature token's driver software (e.g. PKCS#11 driver or Minidriver in Windows environment) must be installed in the user's system
2. The browser signing module must be installed in the user's browser (accessible from https://installer.id.ee/)
3. For security reasons, the signing module can only be used with HTTPS protocol

For example, in case of Estonian eID cards, the smart card drivers and browser signing module can be installed by the user from the following locations:

- latest release version of the software: https://installer.id.ee/
- beta version of the software: http://id.eesti.ee (in Estonian)

3.3 Component model

Overview of the components that are used during signature creation in a web application with an eID card is shown in the figure below. The figure also includes optional DigiDoc components (with blue colour) which are needed for creating signed document in DigiDoc format.

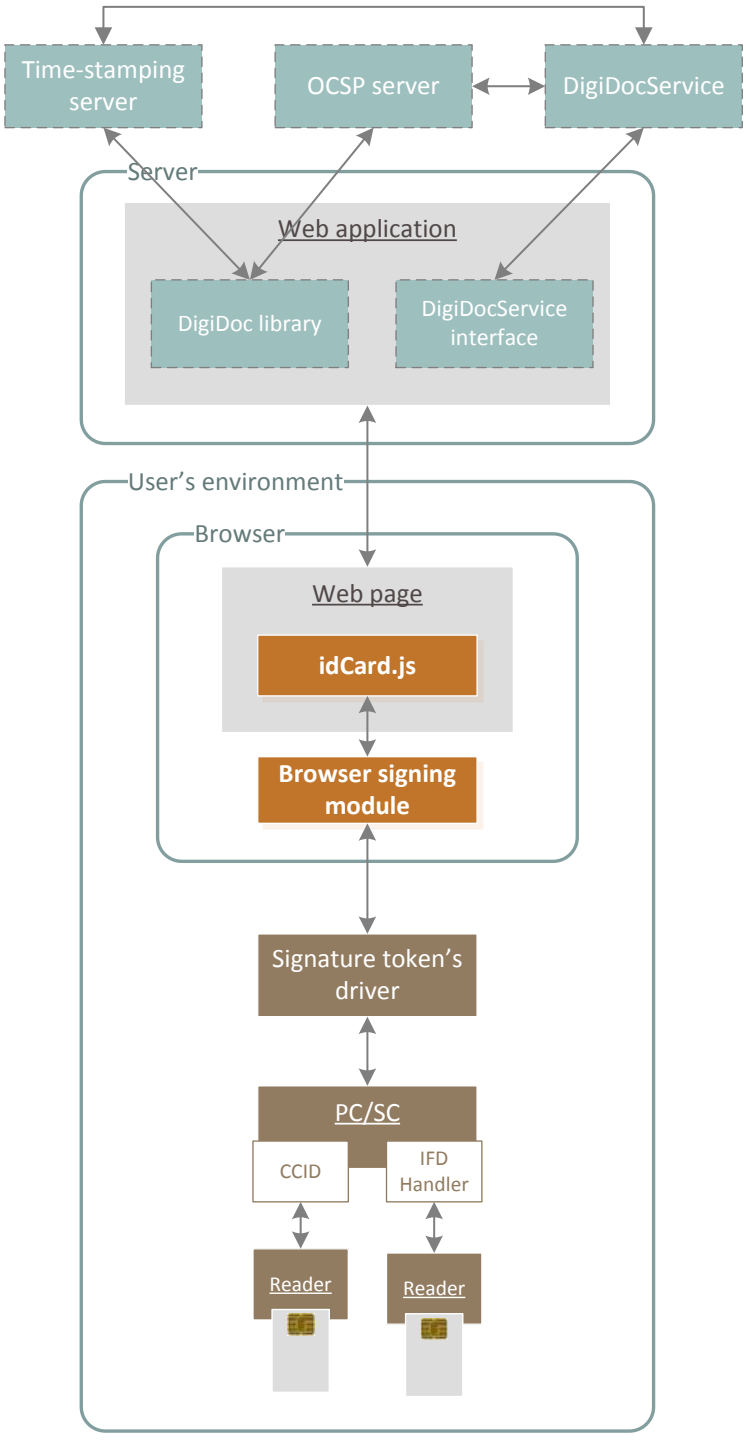

Figure 1. Components used for signing with smart card in browser

The components in figure 1 are described in more detail below. The components are divided into three categories:

1. **Optional components/interfaces in the web application used for creating digitally signed files in DigiDoc format** (with blue colour)

   The web application may be required to create a digitally signed file as a result of the signing process. The digitally signed file format is application context specific and does not directly depend on the idCard.js library or the browser signing module.

   For example, in Estonia, the digital signature formats that are used are BDOC 2.1 [2] and DIGIDOC-XML 1.3 [3] (also called DigiDoc documents; based on XAdES digital signature standard [4]), documents in these formats can be created with **DigiDocService web service** or **DigiDoc software libraries** (or with their combination).

Component	Description
DigiDocService	DigiDocService [5] is SOAP-based web service that enables creating digitally signed documents with eID cards and Mobile-ID and verifying signed documents. Interacts with OCSP server to request validity confirmation of the signer's certificate. Alternatively, DigiDoc libraries (Java, C++ or C) can be used for DigiDoc document creation.
DigiDocService interface	DigiDocService interface is used in this document to denote the component in a web application which interacts with DigiDocService.
DigiDoc library	Software library which is part of DigiDoc system [6], possible options are JdigiDoc (Java) [7], Libdigidocpp (C++) [8] and CdigiDoc © [9]. The library can be used in web applications for composing digitally signed DigiDoc documents in BDOC 2.1 [2] or DigiDoc-XML 1.3 [3] formats. Interacts with OCSP server to request validity confirmation of the signer's certificate. Alternatively, DigiDocService web service can be used for DigiDoc document creation.
OCSP server	Used for requesting validity confirmation of the signer's certificate in the course of digital signature creation process, according to RFC 6960 ([10]). The OCSP server returns the signed response, and the signer's certificate is either approved or rejected, based on whether or not the OCSP server validates the certificate. The OCSP response is incorporated in the digitally signed DigiDoc document.
Time-stamping server	Used for requesting confirmation of the time when the signature value existed, i.e. can be used as a trusted source to determine the time when the signature was created. The time-stamp format has been described in [11].

2. **Components in the web application that are responsible for enabling communication with the user's smart card (with orange colour)**

Component	Description
idCard.js	JavaScript library [1] that simplifies implementing signature creation in web applications (i.e. in browser environment) with Electronic Identity Cards (eID) or other PKI-enabled tokens.

Browser signing module	Multi-platform browser signing module which is used by the idCard.js library. The module is responsible for communication with the signature token's (e.g. ID-card or other token) driver in the signer's computer.

3. Components in the user's operating system that are used for communicating with the signature token (eID card, with brown colour):

Component	Description
Signature token's driver	Driver (e.g. PKCS#11 driver or Minidriver in Windows) to communicate with signature token (e.g. smart card, HSM, USB token) in the signer's environment.
PC/SC	Standard communication interface between the computer and the smart card, a cross-platform API for accessing smart card readers
CCID	USB driver for Chip/Smart Card Interface Devices
IFDHandler	Interface Device Handler for CCID readers
Reader	Device used for communication with a smart card

3.4 Browser signing module

Browser signing module is an intermediate component between the end-user's browser environment and the signature token's driver.

Browser signing module is a common name for signature creation browser plug-in (also referred to as 'digidocPlugin' in the context of idCard.js library) and extension. Implementation of the signing module depends on the user's environment:

1. **EstEID Firefox plug-in**. Supported in Mozilla Firefox, Chrome(*), Internet Explorer, Safari. Data exchange with the plug-in is synchronous.

2. **EstEIDPluginBHO plug-in**. Supported in Internet Explorer. Data exchange with the plug-in is synchronous

3. **chrome-token-signing** browser extension - supported in Chrome(*). Data exchange is asynchronous.
   Note: the chrome-token-signing extension includes JavaScript components that are loaded to every web page that the user opens. The JavaScript components also overwrite some functions and variables of the idCard.js library in order to replace some behaviour that is not needed in case of the extension (but is needed for the plug-in).

(*) Starting from Chrome browser's version M35, the EstEID Firefox plug-in is gradually replaced with chrome-token-signing extension. The replacement starts with Linux environment and continues to other operating systems according to Chrome's schedule for ending support of NPAPI-based plug-ins.

Note that the idCard.js library's user is not required to know details of the signing module's implementation - the idCard.js library is meant to hide the low level details from the web application's developer.

4. Using idCard.js API

The following chapter lists the idCard.js library's API methods in arbitrary order. Sample code for implementing the API calls can be seen in the **sign.html** sample application which is included in the library's distribution package.

NB! The asynchronous getVersion(), getCertificate() and sign() functions must be called out one at a time, i.e. the result of one of the asynchronous functions must be received from the signing module before calling out another asynchronous function.

4.1 General methods and variables

4.1.1 Variable libraryVersion

idCard.js library's version number value is stored in the 'libraryVersion' variable (used since version 0.20 of the library). **Note: it is strongly recommended that the libraryVersion variable's value is logged by the web application in case of each signing operation (or each user session)**. **See also chap. 4.4.**

Note: IdCardPluginHandler object's method 'getVersion(successCallback, failureCallback)' specifies the version of the plug-in, not the library.

4.1.2 Method getType()

The method returns the browser signing module's type, the possible values are:

- SYNC – the low-level communication with the signing module is done synchronously
- ASYNC – the low-level communication with the signing module is done asynchronously.

Note that the idCard.js library supports both types of the signing module. The functions that are marked as asynchronous (described later in this document) must be used asynchronously despite the type of the signing module's type.

Note: it is strongly recommended that the getType() function's return value is logged by the web application in case of each signing operation (or each user session). **See also chap. 4.4.**

4.1.3 Method loadSigningPlugin(lang, pluginToLoad)

The method loads the browser signing module to web page. It is expected that there exists an HTML element with 'id' attribute value "**pluginLocation"** in the web page where the module can be loaded, e.g:

```
<div id="pluginLocation"></div>
```

Note: In order for the signing module to load successfully in Firefox and Chrome browsers, the element should not be used in the following way:

~~ ~~

Parameters:

Name	Data type	Description
lang	String	Sets the language of returned error messages. Possible values: ‘est’, ‘eng’, ‘rus’
pluginToLoad	String	Optional. Possible value is ‘digidocPlugin’

4.2 Signature creation

4.2.1 Object IdCardPluginHandler(lang)

Offers functionality for reading signature certificate from the signature token and calling out the signature creation operation on the token.

Parameters:

Name	Data type	Description
lang	String	Sets the language of returned error messages. Possible values: ‘est’, ‘eng’, ‘rus’

NB! Since version 0.20 of the idCard.js library, the API of **IdCardPluginHandler** methods and their usage principles have been changed due to changes in the browser signing module – usage of the methods is now **asynchronous**.

Users of the library’s version 0.14 and earlier must change getVersion(), getCertificate() and sign() methods’ API usage in order to use v0.20 of the library. **Detailed description is provided in the following sections.**

4.2.1.1 Method getVersion(successCallback, failureCallback)

The getVersion() method returns asynchronously the version of the browser signing module that is used by the idCard.js library.

Note: it is strongly recommended that the getVersion() method’s return value is logged by the web application in case of each signing operation (or each user session). **See also chap. 4.4.**

Parameters:

Name	Data type	Description
successCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to **asynchronously** return the result of getVersion() method in case of **successful scenario**. The returned value is the version number (as String).
failureCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to **asynchronously** return the result of getVersion() method in case of **failure scenario**. The returned value is an IdCardException object (containing the error message and code).

*** Parameter names of getVersion() method have to be set by the library's user. The chosen names determine methods which are called by the library to return the getVersion() method's result asynchronously.**

Sample:

```
<!–call out getVersion() method →
new IdCardPluginHandler('est').getVersion(handleVersion, handleError);

<!–implement method which is called out in case of successful scenario→
function handleVersion(version) {
        <!–implement code for using the signature value →
}

<!–implement method which is called out in case of failure scenario→
function handleError(ex) { <!--implement code to handle the error situation --> }
```

Description:

- to receive the result of successful scenario of the getVersion() method (the version number), the library's user must implement a method and name it according to the value that s/he set to the "**successCallback**" parameter. The method must have an input parameter of type String. The library calls out this method if signing succeeds.
- in case of failure in the process of signing, the library calls out method which is determined by the value of "**failureCallback**" input parameter. The library's user must implement this method, the method must have an input parameter of type idCardException.

See also **sign.html** sample application's source code, the sample is included in the idCard.js library's distribution package.

Note: idCard.js library's variable 'libraryVersion' specifies the version of the library, not the browser signing module.

4.2.1.2 Method getCertificate(successCallback, failureCallback)

The method returns asynchronously the signer's certificate. In case of an error situation, IdCardException object is returned

Parameters:

Name	Data type	Description
successCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to **asynchronously** return the result of getCertificate() method in case of **successful scenario**. The returned value is a Certificate object (the signer's certificate from the smart card).
failureCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to **asynchronously** return the result of getCertificate() method in case of **failure scenario**. The returned value is an IdCardException object (containing the error message and code).

*** Parameter names of getCertificate() method have to be set by the library's user. The chosen names determine methods which are called by the library to return the getCertificate() method's result asynchronously.**

Sample:

```
<!–call out getCertificate() method →
new IdCardPluginHandler('est').getCertificate(handleCertificate, handleError);

<!–implement method which is called out in case of successful scenario→
function handleCertificate(cert) {
        <!–implement code for using the certificate -->
}

<!–implement method which is called out in case of failure scenario→
function handleError(ex) {
        <!–implement code to handle the error situation -->
}
```

Description:

- to receive the result of successful scenario of the getCertificate() method (the signer's certificate), the library's user must implement a method and name it according to the value that s/he set to the "**successCallback**" parameter. The method must have an input parameter of type Certificate. The library calls out this method if reading the signer's certificate succeeds.
- in case of failure in the process of reading the certificate from card, the library calls out method which is determined by the value of "**failureCallback**" input parameter. The library's user must implement this method, the method must have an input parameter of type idCardException.

See also **sign.html** sample application's source code, the sample is included in the idCard.js library's distribution package.

4.2.1.3 Method sign(id, hash, successCallback, failureCallback)

Method for signing the hash value with the signer's private key (in the smart card). Before using this method, the signing certificate's id value has to be determined by using getCertificate() method.

NB! Sign() method should not be called out before the getCertificate() method's results have been returned by the library. For example, the "Sign" button can be enabled for the end user only after the getCertificate() has returned the signer's certificate.

Parameters:

Name	Data type	Description
id	String	Identifier of the certificate which is used for signing in the card. The identifier must be determined with method getCertificate() (identifier is the id attribute of the returned Certificate object).
Hash	String	The hash to be signed in hexadecimal format, e.g „FAFA0101FAFA0101FAFA0101FAFA0101FAFA0101“
successCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to

		asynchronously return the result of sign() method in case of **successful scenario**. The returned value is the signature value (as hexadecimal String).
failureCallback*	String	Value of this parameter has to be set to the name of the method which will be used by the library to **asynchronously** return the result of sign() method in case of **failure scenario**. The returned value is an IdCardException object (containing the error message and code).

*** Parameter names of sign() method have to be set by the library's user. The chosen names determine methods which are called by the library to return the sign() method's result asynchronously.**

Sample:

```
<!–call out sign() method -->
new IdCardPluginHandler('est').sign(id, hash, handleSignature, handleError);

<!–implement method which is called out in case of successful scenario→
function handleSignature(signature)  {
        <!–implement code for using the signature value -->
}

<!–implement method which is called out in case of failure scenario→
function handleError(ex) {
        <!–implement code to handle the error situation -->
}
```

Description:

- to receive the result of successful scenario of the sign() method (the signature value), the library's user must implement a method and name it according to the value that s/he set to the "**successCallback**" parameter. The method must have an input parameter of type String. The library calls out this method if signing succeeds.
- in case of failure in the process of signing, the library calls out method which is determined by the value of "**failureCallback**" input parameter. The library's user must implement this method, the method must have an input parameter of type idCardException.

See also **sign.html** sample application's source code, the sample is included in the idCard.js library's distribution package.

4.2.2 Object Certificate

Structure which is used to store the signer certificate's data. The Certificate object is returned asynchronously by IdCardPluginHandler object's method getCertificate().

Certificate structure's fields are as follows:

Name	Data type	Description
id	String	Certificate's identifier
cert	String	Certificate in hexadecimal format

CN	String	The certificate's "Subject" field's Common Name (CN) value.
issuerCN	String	The certificate's "Issuer" field's Common Name (CN) value.
validFrom	String	Start time of the certificate's validity period, Zulu time zone, in format „dd.mm.yyyy hh:mm:ss"
validTo	String	End time of the certificate's validity period, Zulu time zone, in format „dd.mm.yyyy hh:mm:ss"

4.3 Exception handling

4.3.1 Object IdCardException

In case of error situation, the IdCardException object is returned.

IdCardException structure's fields are as follows:

Name	Data type	Description
returnCode	String	Error code value
message	String	Error message

The possible error codes (values of returnCode) and the respective error messages are as follows:

Error code	Meaning
1	Signing was cancelled
2	Certificate not found
9	Incorrect PIN code
12	Unable to read ID-Card
14	Technical error
15	Unable to find software
16	Invalid certificate identifier
17	Invalid hash
19	Web signing is allowed only from https:// URL
100	Web signing module is missing from your computer or web signing is not supported on your operating system and browser platform. Signing software is available from https://installer.id.ee

4.3.1.1 Method isCancelled()

isCancelled() returns 'true' if the user cancelled the signing process (i.e. the error code is 1).

4.3.1.2 Method isError()

isError() returns 'true' in case of all error codes except of 0 and 1.

4.4 Logging idCard.js usage

To enable solving any possible error situations with the created signatures in the future then it is strongly recommended that the following data are logged by the web application in case of each signing operation (or each user session):

1. The signing module's version number – can be read with IdCardPluginHandler.getVersion(successCallback, failureCallback) method.
2. The signing module's type (SYNC/ASYNC) – can be read with getType() method.
3. The idCard.js library's version number – can be read from 'libraryVersion' variable.
4. User agent data.

For example (in PHP), log the data in only once during a session:

```
if (!isset($_SESSION["is_plugin_version_recorded"])) {
	debug_log("Module version: " . getPost("moduleVersion") . "; Module type:
	" . getPost("moduleType") . "; idCard.js version: " .
	getPost("libraryVersion") . "; Useragent: " .
	$_SERVER['HTTP_USER_AGENT']);

	$_SESSION["is_plugin_version_recorded"] = 1;
}
```

5. References and additional resources

[1] idCard.js library	JavaScript client library for simplifying signing in web applications http://id.ee/index.php?id=30496
[2] BDOC2.1:2013	BDOC – Format for Digital Signatures. Version 2.0:2013 https://www.sk.ee/repository/bdoc-spec21.pdf http://id.ee/public/bdoc-spec21-est.pdf
[3] DigiDoc format	DigiDoc file format http://id.ee/public/DigiDoc_format_1.3.pdf
[4] XAdES	ETSI TS 101 903 V1.4.2 (2010-12) – XML Advanced Electronic Signatures http://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf
[5] DigiDocService	DigiDocService – what is it? http://www.id.ee/?id=35785
[6] DigiDoc libraries	http://id.ee/index.php?id=30486
[7] JDigiDoc	DigiDoc Java library - JDigiDoc http://id.ee/index.php?id=35783
[8] Libdigidocpp	DigiDoc C++ library - Libdigidocpp http://id.ee/index.php?id=36484
[9] CDigiDoc	DigiDoc C library - CDigiDoc http://id.ee/index.php?id=35782
[10] RFC6960	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP http://tools.ietf.org/html/rfc6960
[11] RFC3161	Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) http://tools.ietf.org/html/rfc3161

6. Terms and acronyms

BDOC 2.1 (.bdoc)	Term is used to denote a digitally signed file format which is a profile of XAdES and follows container packaging rules based on OpenDocument and ASiC standards. The document format has been defined in [2].

chrome-token-signing	Browser extension for digital signing, used by the idCard.js library. The extension is responsible for communication with the signature token's (e.g. ID-card or other token) driver in the signer's computer, it is used in Chrome browser when the digidocPlugin is not supported.
digidocPlugin	Multi-platform digital signing browser plug-in which is used by the idCard.js library. The plug-in is responsible for communication with the signature token's (e.g. ID-card or other token) driver in the signer's computer. In Chrome browser, when digidocPlugin is not supported then chrome-token-signing extension is used instead.
DigiDocService (DDS)	DigiDocService [5] is SOAP-based web service which enables creating digitally signed documents with eID cards and Mobile-ID and verifying signed documents. Interacts with OCSP server to request validity confirmation of the signer's certificate.
DigiDoc library	Software library which is part of DigiDoc system [6]. Libraries which support signature creation are JDigiDoc (Java) [7], Libdigidocpp (C++) [8] and CdigiDoc (C) [9]. The library can be used in web applications for composing digitally signed DigiDoc documents in BDOC 2.1 [2] or DigiDoc-XML 1.3 [3] formats. Interacts with OCSP server (and optionally time-stamping server) to request validity confirmation of the signer's certificate.
DIGIDOC-XML (.ddoc)	The term is used to denote a DigiDoc document format that is based on the XAdES standard and is a profile of that standard. The current version is 1.3 which has been described in [3].
OCSP	Online Certificate Status Protocol, an Internet protocol used for obtaining the revocation status of an X.509 digital certificate
OCSP Responder	OCSP Server, maintains a store of CA-published CRLs and an up-to-date list of valid and invalid certificates. After the OCSP responder receives a validation request (typically an HTTP or HTTPS transmission), the OCSP responder either validates the status of the certificate using its own authentication database or calls upon the OCSP responder that originally issued the certificate to validate the request. After formulating a response, the OCSP responder returns the signed response, and the original certificate is either approved or rejected, based on whether or not the OCSP responder validates the certificate.
SK	AS Sertifitseerimiskeskus (Certification Centre Ltd.). Certificate Authority in Estonia
XAdES	XML Advanced Electronic Signatures, a set of extensions to XML-DSIG recommendation making it suitable for advanced electronic signature. The signature format has been described in [4].

Appendix 1. Signature creation process

Signature creation in web browser includes communication between the web application and the user's signing token (via the idCard.js library and the browser signing module):

1. request the signing certificate from the signing token
2. conduct the signing operation in the token (send the hash to be signed to the token and receive signature value).

Note that the process should also include logging as described in chap. 4.4.

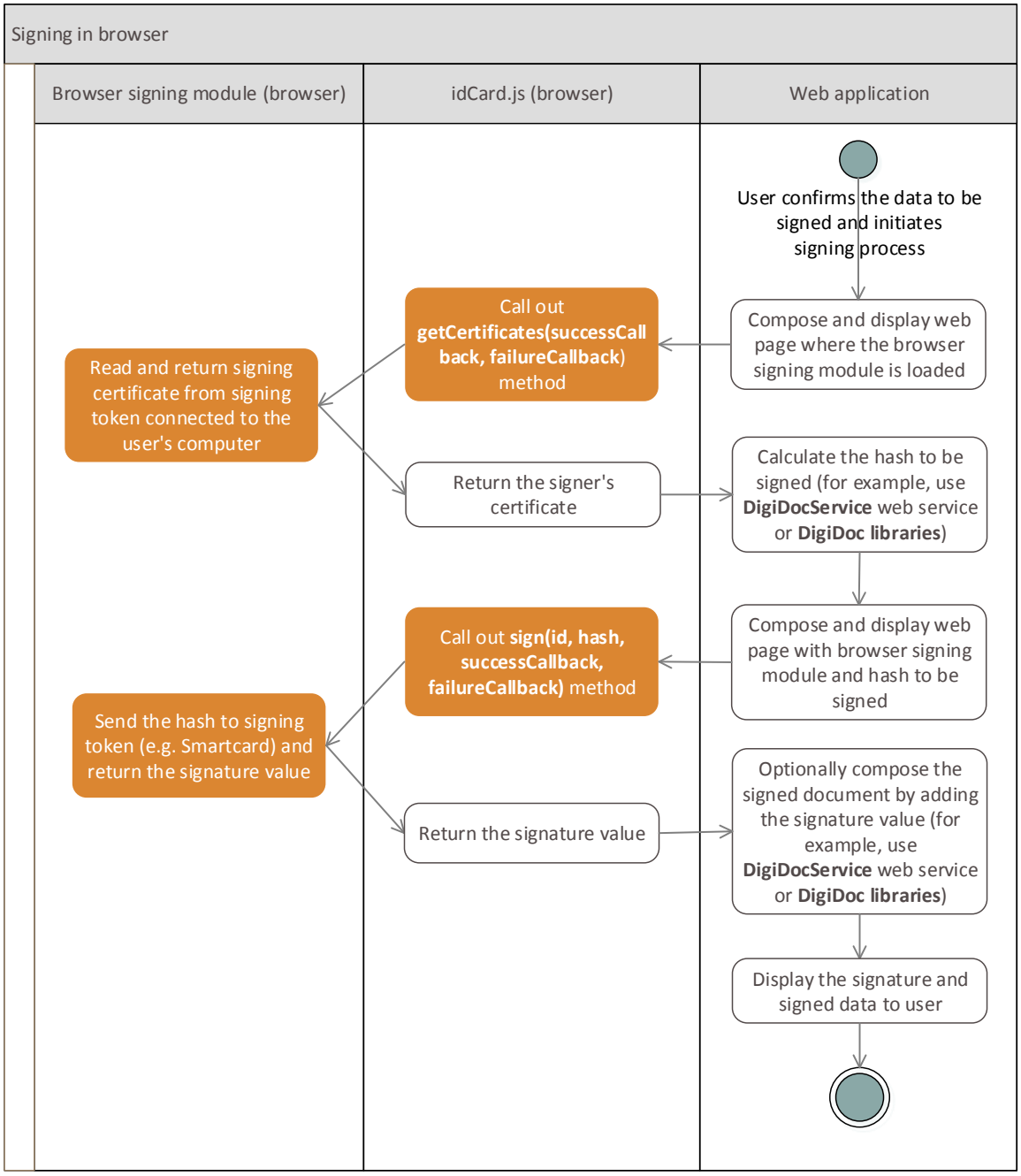

Figure 2. Sample signature creation process in browser